Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--14-9502

Geospatial Analysis and Model Evaluation Software (GAMES): Integrated Web-Based Analysis and Visualization

TIMOTHY R. KEEN
JAMES D. DYKES

Ocean Dynamics and Prediction Branch
Oceanography Division

April 11, 2014

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
11-04-2014	Memorandum Report	

4. TITLE AND SUBTITLE

Geospatial Analysis and Model Evaluation Software (GAMES): Integrated Web-Based Analysis and Visualization

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0602435N

6. AUTHOR(S)

Timothy R. Keen and James D. Dykes

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
73-6669-04-5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Oceanography Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/7320--14-9502

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
One Liberty Center
875 North Randolph Street, Suite 1425
Arlington, VA 22203-1995

10. SPONSOR / MONITOR'S ACRONYM(S)

ONR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The software design methodology discussed in this report is fundamental to Geospatial Analysis and Model Evaluation Software (GAMES) module development. This report describes methods for the development of Web-based applications for accessing, processing, and distributing observations and model simulation results. The primary components discussed in this report include data visualization using Google Earth, geographical analysis using ArcGIS components within an oceanographic framework, and executable scripts designed to access large databases and run user-specified analyses. These components are illustrated using local examples in conjunction with local GServer web server. An example is also given of accessing a global ocean model database and using the results to calculate model statistics and long-term trajectories for surface flotsam in the Pacific Ocean. The components described in this report can be used for either component-based software engineering (CBSE) or service-oriented software engineering (SOSE) designs. The resulting software is applicable to research and operational oceanography.

15. SUBJECT TERMS

Google earth
ArcGIS
Geographical information systems
Web service
Data mining
Global NCOM

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy Keen
Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	109	19b. TELEPHONE NUMBER *(include area code)* (228) 688-4950

Table of Contents

1. Introduction

This report describes a software engineering effort in support of multiple projects in the Nearshore and Coupled Model Systems section (Code 7322), Naval Research Laboratory Stennis Space Center (NRLSSC). The primary focus is on the development of a Web-based application for accessing, processing, and distributing the observations and model simulation results for the 6.1 project, "Transport and mixing of terrigeneous material in the coastal ocean". This work is developing innovative ways to simulate and analyze hydrodynamic mixing processes in estuaries and the nearshore. Additional support for this software comes from the 6.4 effort, "ASW Reach-back Cell Oceanographic Analysis System (ARCOAS)". The incorporation of the resulting software thus has application to both basic research within NRL and applied technology at the Naval Oceanographic Office (NAVOCEANO).

Background

Web-based services (WS) have evolved through several waves since the World Wide Web (WWW or Web) was invented in 1989. The first wave comprised user interfaces to systems that could not be easily accessed over a Local Area Network (LAN) or a Wide Area Network (WAN) (Putz 1994). The second wave used basic Web technologies, e.g., HyperText Transfer Protocol (HTTP) and eXtensible Markup Language (XML), and robust protocols (WS-*) for implementing application programming interfaces (APIs) and business-class (i.e., enterprise class) composite applications (Felber et al. 2003; Gordon 2002). A more recent third wave uses lighter-weight protocols and ad-hoc design approaches to merge or mash-up information or services for use primarily by individuals (Nachouki and Quafafou 2011).

The Hypertext Markup Language (HTML) and its ability to render forms has driven the first wave of WS, which mostly provided Web forms for submitting data to some process being driven by a Web-based front end (England et al. 1996). XML then introduced the ability to represent application-defined data structures. XML is the most important foundation of almost all technologies in WS (El-Bakry et al. 2010). Web-based technologies often bind together heterogeneous applications (Pullen et al. 2005). It is necessary to integrate Web technologies into an application for user or machine access via the Web; however, rather than implementing various Web technologies as an integral part of an application, it is often advantageous to let specialized components (like a Web server) handle the mechanics of Web interactions (Adala and Tabbane 2010; Li et al. 2007). The Hypertext Preprocessor (PHP) is a scripting language that can be used on the server side to process HTTP requests and generate dynamic web pages (Pais et al. 2010). PHP can access other web-based services by accessing them through various interfaces (Bittencourt et al. 2008) and it can also be used to implement Web-based services by returning XML instead of HTML (Villoldo et al. 2007). This means that PHP can be used as a simple platform for WS, both for the client and for the server side (Wang et al. 2001).

Middleware is a classical component of large heterogeneous distributed systems (Carone 1996; Malek et al. 2010; Vandervoort 1992). The history of middleware is rooted in software engineering, based on the observation that programming abstractions can do two important things at the same time: (1) make programming more efficient by providing support for tasks that have to be solved repeatedly; (2) provide abstractions that hide heterogeneity, so that

Manuscript approved February 18, 2014.

applications using these abstractions can communicate regardless of differences on lower layers.

Web-based services can be invoked using Web technologies, and provide results in some format. Web-oriented services often provide HTML as the result format, which is not well suited for processing. Application-oriented services often provide XML as the result format, which is easier to process and can be structured specifically for the needs of a service. One of the key questions when designing and using XML-based services is the underlying data model of the XML (Osipov et al. 2000; Ravat et al. 2010). Web Services (the narrowly defined field as opposed to the broader topic of WS) are usually described as three technologies (Curbera et al. 2002). The Simple Object Access Protocol (SOAP) is used to encapsulate XML payload and provides a common format that can be interpreted and processed by all SOAP-aware applications (Liu et al. 2001). The Web Services Description Language (WSDL) is used to describe the interface to a Web service which accepts SOAP messages (Anonymous 2001). The Universal Description, Discovery, and Integration (UDDI) format is a registry that can be used to search for WSDL documents (Anonymous 2000).

An API is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API (Maaref et al. 1997; Weis 1996). It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. An API may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer program and the implementer program of the API. When used in the context of Web development, the API (Web API) is typically a defined set of HTTP request messages and a definition of the structure of response messages, which is usually in an XML or JavaScript Object Notation (JSON) format (Kono et al. 2009).

Whereas Web API is virtually a synonym for Web service, the recent trend is away from SOAP-based services towards more direct Representational State Transfer (REST) (Fielding and Taylor 2002) style communications (Battle and Benson 2008; Khare et al. 2004; Kwon et al. 2009). REST is an abstraction of the architectural elements within a distributed hypermedia system. Three classes of elements can be identified: components, or processing elements; data elements; and connecting elements. REST ignores the details of component implementation and protocol syntax and focuses on the roles of components, how their interaction with other components is constrained, and how they interpret significant data elements (Fielding and Taylor 2002). This focus results in the principle difference between REST and other network-based architectures. It emphasizes a uniform interface between components. This generality reduces the overall system complexity though it decreases efficiency.

The POSIX (Portable Operating System Inteface for Unix) standard defines an API that allows a wide range of common computing functions to be written such that they may operate on different systems, such as MacOS X, and various Berkeley Software Distributions (BSDs) (Leathrum and Liburdy 1995; Severance 1999); however, making use of this requires re-compiling for each platform. A compatible API, on the other hand, allows compiled object code to function without any changes to the system implementing that API (Rezende et al. 2009). This is beneficial to software providers because they may distribute existing software on new systems without producing and distributing upgrades. Users may install older software on their

new systems without purchasing upgrades, although this generally requires that various software libraries implement the necessary APIs as well.

A plug-in (or plugin) is a set of software components that adds specific capabilities to a larger software application (Mezini and Lieberherr 1998). The host application provides services that the plug-in can use, including registration and a protocol for data exchange. Plug-ins depend on the services provided by the host application and do not usually work by themselves. Conversely, the host application operates independently of the plug-ins, making it possible for end-users to add and update plug-ins dynamically without needing to make changes to the host application. Open APIs provide a standard interface, allowing third parties to create plug-ins that interact with the host application (Poo and Chew 1996; Sousa and Freitas 1998). A stable API allows third-party plug-ins to continue to function as the original version changes and to extend the life-cycle of obsolete applications (Mclellan et al. 1998). Software developers can implement plug-in frameworks using libraries written in several computer languages: (1) Pugg and OFX (C++); (2) NetBeans Platform and Java Plug-in Framework (Java); (3) twisted and PyUtilib (Python); and (4) .NET and Compact Plugs (.NET). Applications support plug-ins for several reasons. Plugins allow third-party developers to create capabilities that extend an application and thus support new features. They also help to reduce the size of an application and separate source code from applications that may have incompatible software licenses.

Specific examples of applications that use plug-ins:

- Email clients use plug-ins to decrypt and encrypt email.
- Graphics software uses plug-ins to support file formats and process images.
- Media players use plug-ins to support file formats and apply filters.
- Microsoft Office uses plug-ins (better known as add-ins) to extend the capabilities of its application by adding custom commands and specialized features.
- Packet sniffers use plug-ins to decode packet formats.
- Remote sensing applications use plug-ins to process data from different sensor types.
- Audio spectrum analysis applications can accept plug-ins for third-party digital signal processors.
- Software development environments use plug-ins to support programming languages.
- Web browsers use plug-ins (often implementing the NPAPI specification) to play video and presentation formats and a plethora of other things e. g. Adobe Acrobat, Microsoft Office, Sivler Light, DBsign Web Signer, etc.

Designing and implementing a large and complex software system is a difficult task. Two well-recognized software engineering paradigms have been developed to deal with this challenge (Figure 1.1): (1) component-based software engineering (CBSE)(Crnkovic, 2001); and (2)

service-oriented software engineering (SOSE) (Breivold and Larsson 2007). CBSE is an established approach that supports building systems for distributed and web-based applications through the composition and assembly of software components. CBSE facilitates management of complexity and increases usability while shortening the time to market. The requirements of Web computing and network-based applications and systems are motivators for the development of SOSE applications, which utilize services as fundamental elements in software solutions. These paradigms have continued development along parallel tracks but with different focus. Their approaches and technologies are similar because SOSE evolved from CBSE frameworks (Tsai 2005). These similarities can lead to confusion in applying them effectively.

Component models specify the standards that components need to follow during composition and interaction. Current research topics in CBSE include Quality of Service (QoS), error predictability, component interference and classification, identification and selection, adaptation, and testing and deployment techniques. As suggested by this list of current research topics, CBSE does not address all the complexities associated with varying platforms, protocols, and devices, as well as Web-based issues. A key concept is that components are the building blocks of CBSE. The components can be deployed independently and may be composed by third parties (Crnkovic et al. 2007; Crnkovic and Larsson 2002). This necessitates a degree of modularization that allows for evolution of components that have independent development paths. The individual components are assembled using glue code to form an application. The composition is made from several component instances that are connected and interact together. Components must conform to rules that are established by a specific component model. Consequently, the practicality of incorporating components of different component models is relatively limited; however, compliance to a certain technology may also lead to advantages in optimization of solutions by the specific technology. Another result of this paradigm is that the degree of encapsulation is flexible, from a black box to a white box that exposes all of the implementation.

SOSE, or Service Oriented Architecture (SOA), implementations rely on a mesh of software services (Fountain 2003). Services are the fundamental elements for developing applications and solutions (Papazoglou and Van Den Heuvel 2007) using a range of elements (Figure 1.2). This paradigm supports the development of rapid, low-cost, and easy composition of distributed applications in heterogeneous environments. Services comprise unassociated, loosely coupled units of functionality that have no calls to each other embedded within them. Each service implements one action, such as filling out an online application for an account, or viewing an online bank statement, or placing an online booking or airline ticket order. Rather than services embedding calls to each other in their source code, they use defined protocols that describe how services pass and parse messages using description metadata. SOA developers associate individual SOA objects by using orchestration and choreography (Peltz 2003). In the process of orchestration the developer associates software functionality (the services) in a non-hierarchical arrangement using a software tool that contains a complete list of all available services, their characteristics, and the means to build an application utilizing these sources. The composite service is built by composing service descriptions and the realization of the service composition is during run time when the service providers are discovered and bound (Breivold and Larsson 2007).

Service-based applications have three main parts: (1) a provider; (2) a consumer; and (3) a registry (Huhns and Singh 2005). A minimal definition of a service can include the following two criteria (Parastatidis et al. 2005):

- A service is the logical manifestation of some physical or logical resources (e.g., databases, programs, devices, humans) and/or some application logic that is exposed to the network;
- Service interaction is facilitated by message exchanges.

The great promise of SOA suggests that the marginal cost of creating a new application is low because all of the required software already exists to meet the requirements of other applications. Ideally, one requires only orchestration to produce a new application (Stojanovic et al. 2004). SOA features loose coupling, in contrast to the functions that a linker binds together to form an executable, or to a dynamically linked library, or to an assembly. SOA services also run in safe wrappers (such as Java or .NET) and other programming languages that manage memory allocation and reclamation, allow ad hoc and late binding, and provide some degree of indeterminate data typing. These ideas are represented by a number of design principles (Breivold and Larsson 2007; Parastatidis et al. 2005):

- Boundaries are explicit. The boundaries of a service are well defined and other services do not see the internal workings, implantation details, or resource representations of a service.
- Services are autonomous. Service implementations are developed and evolve independently from one another.
- Services share schema, not classes. In SOAs, no single set of implementation-specific abstractions (classes) spans an entire application. Instead, services share contracts that define the structure of the information that they exchange.
- Policies determine service compatibility. Services interact with one another only after it has been determined that they can meaningfully exchange information

Web services can implement either component-based or service-oriented architecture. Web services make functional building-blocks accessible over standard Internet protocols independent of platforms and programming languages (Sacha et al. 2010). These services can represent either new applications or just wrappers around existing legacy systems to make them network-enabled (Qu et al. 2007). Implementations can use one or more of these protocols and, for example, might use a file-system mechanism to communicate data conforming to a defined interface-specification between processes conforming to the SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks.

These principles can be refined into the following key concepts for software development in modern applications (Bennett et al. 2001):

- meet necessary and sufficient requirements—overengineered systems are not required;
- personalization—tailored to individual user's requirements;
- self-adapting—reflective processes are required to monitor use and implement changes to meet user requirements, interface styles, and work patterns;
- fine-grained—small, simple units that cooperate through rich communication structures, thus providing resistance to failure and facilitating self-adaptation and personalization;
- transparency—viewed as a single abstract object even when distributed across platforms and geography.

These principles are necessary to develop demand-led software services, which are required for emergent organizations with continual change (i.e., engineering for emergent solutions).

Web services are required to satisfy high quality standards and automated test tools are necessary to improve quality of services (Bertolino and Acm 2009; De Melo and Silveira 2010). Such tools for headless services (including message and database services along with web services) are much more difficult to implement. The REST (Representational State Transfer) architectural model (Figure 1.3) has been used with some success (Chakrabarti et al. 2009). Another challenge relates to providing appropriate levels of security (Li et al. 2009). Security models built into an application may no longer suffice when an application exposes its capabilities as services that can be used by other applications (Kovac and Trcek 2009; Michelsen and Ieee 2007).

The term "Web 2.0" is associated with web applications that combine mature implementation techniques, the mashing of existing Web services into value-added applications, and emphasis on community and collaboration, as well as the establishment of direct distribution platforms for user-generated content (Boll 2007). A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as consumers of user-generated content in a virtual community (Lin 2007). The client-side/web browser technologies used in Web 2.0 development are Asynchronous JavaScript and XML (Ajax), Adobe Flash and the Adobe Flex framework, and JavaScript/Ajax frameworks. Multiple data sources are accessed more easily with technologies like XML Remote Procedure Call (XML-RPC), REST, RSS (i.e., Really Simple Syndication), Atom, and mashups, which facilitate the subscription, propagation, reuse, and intermixing of Web content.

Web 2.0 can be distinguished from the classical Web by three characteristics (Ankolekar et al. 2008):

- Community – Contributors collaborate and share information easily, which produces an emerging result that could not be achieved by each individual.

- Mashups – Data from different sites are pulled together to provide new values, thus allowing handcrafted products merged from disparate sources.

- Ajax – The technology contained in this methodology has enabled responsive user interfaces that allow the other components to work properly.

Web APIs allow the combination of multiple services into new applications known as mashups, which are Web applications developed using contents and services available online (Yu et al. 2008). Mashups typically serve a specific (short-lived) need and are composed of easy-to-use Web technologies. Generally, however, integrating enterprise data and applications into a useful mashup requires programming skill and knowledge about the databases used in the composite application. Several tools have been developed to assist users in developing their own mashups: (1) Yahoo Pipes (http://pipes.yahoo.com); (2) Google Mashup Editor (http://editor.google.mashups.com); (3) Microsoft Popfly (www.popfly.ms); (4) Intel Mash Maker (http://mashmaker.intel.com); and (5) Quick and Easily Done Wiki (http://services.alphaworks.ibm.com/qedwiki). These mashup tools are based on a browser whereas traditional integration technologies offer desktop applications.

The mashup paradigm comprises the objects of integration (components) and how they are glued together (composition logic). The Component Model (CM) can be characterized by type, interface, and extensibility. Type can be data, application logic, or user interface. The interface can consist of combinations of the create-read-update-delete (CRUD) interface and GUI elements exposed to the user. Extensibility permits the user to create new components or extend the CM to new requirements. The Composition Model (CoM) determines how components are integrated to form the mashup. The CoM output type can be data, application logic, or user interface, depending on whether the composition provides data, programmable APIs, or user interfaces. The orchestration style (definition and synchronization of components' execution) can be defined as flow-based, event-based, or layout-based. Data passing can be by dataflow or blackboard (i.e., variables) approaches. Finally, a composition can be instance-based or continuous. Mashups are deployed in a standalone fashion on a Web server that may be managed by the developer or a third-party (e.g., Google). A mashup can be assembled on a server via PHP or Ruby or on the client, typically using JavaScript inside a Web browser. Mashups are about simplicity, usability, and ease of access. These requirements preclude feature completeness or extensibility. Further evolution of the mashup paradigm depends on the future availability of a user-interface component model, middleware for user interface integration, and the identification of appropriate component and composition models.

Web 2.0 draws together the capabilities of client- and server-side software, content syndication and the use of network protocols. Web 2.0 sites provide users with information storage, creation, and dissemination capabilities that were not possible in the environment now known as "Web 1.0" or the classical Web. There are serious challenges in the information quality and security for Web 2.0 applications, however, because of a lack of editorial control over user-uploaded data (Almeida et al. 2010; Davidson and Yoran 2007).

The complement to Web 2.0 is the Semantic Web (Greaves and Mika 2008), which has quietly evolved from the need for integrated and uniform access to information sources and services as well as intelligent applications for information processing (Decker et al. 2000; Hellman 1999; Nachouki and Quafafou 2011; Staab et al. 2000; Trillo et al. 2010). The Semantic Web was developed in response to the need to identify a service's capability sufficiently to permit its discovery, deployment, composition, and synthesis (Kona et al. 2007) (Figure 1.4). This approach includes methods utilizing USDL (Unified Service Description Language), OWL-S (Semantic Web Ontology Language), WSML (Web Service Modeling Language), and WSDL-S (Semantic Web Service Description Language). The discovery of available services through a repository (service registry) is fundamental if a query is to be answered using a composition of several services.

An underlying concept for the Semantic Web is an ontology (Community Ontology in Figure 1.4), which is defined as a formal and explicit specification of a shared conceptualization (Medjahed and Bouguettaya 2005). Ontologies are critical to enabling semantics-driven data access and processing. Ontology is easily expanded to the concept of a community of Web services, which can be used to define a metadata ontology, which allows the description of other concepts within the community. Ultimately, communities are created by community providers (e.g., government agencies or businesses), which use the community ontology as a template. These communities are published in a registry (e.g., UDDI) so that they can be discovered by service providers. The Internet Reasoning Service (IRS-III) is an example of an execution environment that enables semantic descriptions of a deployed Web service to be used during discovery, mediation, composition, and invocation activities (Dietze et al. 2007). Mediation is the definitive activity of IRS-III because it mediates between a service requester and service providers. This kind of broker utilizes a SWS Library, where semantic descriptions of WS, and the reference Domain Ontologies and Knowledge Bases (instances) are stored.

Objectives

This software engineering effort supports both basic and applied applications, which could be approached with divergent existing technologies. The intent of this effort, therefore, is to identify and demonstrate Web-based approaches that can be implemented either in a restricted environment or openly on the Web. Some aspects of the work are motivated by the objectives of the ARCOAS (ASW Reach-back Cell Oceanographic Analysis System) project, which must mesh with existing and planned technologies at NAVOCEANO. Web-based applications are in their infancy at NAVOCEANO, and this makes the task of selecting available Web approaches more flexible. This freedom will be explored in this work as it pertains to the development of the software architecture model (e.g., component-based or SOA).

The applied side of this effort is a continuation of the ARCOAS software. ARCOAS is a component-based desktop application with limited Web implementation through its use of the ArcMap™ application from ESRI (Environmental Systems Research Institute; http://www.esri.com). The core analysis and mapping functions are executed on the client using a local ArcGIS™ license. Supplemental add-in functions have been written in C# which was used to build customizations of ArcMap™ resulting in a tailored desktop application. A goal under the ARCOAS project is enhancing the Web-based functionality of the ARCOAS application. Thus, we include these objectives into the work discussed in this report. These

include implementation of a data model specified by NAVOCEANO, a WS database client as specified by NAVOCEANO, and access to external WS that will supply both GIS and other types of data (e.g., observations). The WS client will be developed in compliance with the Open Geospatial Consortium (OGC) requirements (Bulatovic et al. 2010; Mccreedy et al. 2009).

The motivation of this work from the basic research side is in the display and analysis of ocean model results and observations in a Web browser application. The objectives are to implement both Google Earth and ArcGis Explorer™ applications in the framework of a browser in order to facilitate storing, accessing, and analyizing disparate data types in a user-friendly format. A fundamental objective is thus to migrate data access through a Web server instead of mounted file systems. This work will examine the applicability of both component-based and SOA technologies to this purpose.

The basic and applied aspects of this work have an unknown degree of overlap with respect to the use of Web-based versus desktop applications to achieve these objectives. The overriding principle that will steer this work is the development of common software (e.g., middleware) that can be used in both component-based and Web applications, and the potential to use both Web 2.0 and Semantic Web Services (SWS) approaches in both the secure operational environment at NAVOCEANO, as well as the less-restrictive research environment at NRL.

Table 1.1

Network Link to Remote URL:	• KML data is updated frequently by the author • KML data is less than 2-3 megabytes • KML data is not accessed very often
N**etwork Link to local file on hard drive:**	• KML data is rarely updated • KML data is larger than 2-3 megabytes • KML data is frequently accessed
Network Link Not Necessary:	• KML data is very small (less than 100K)

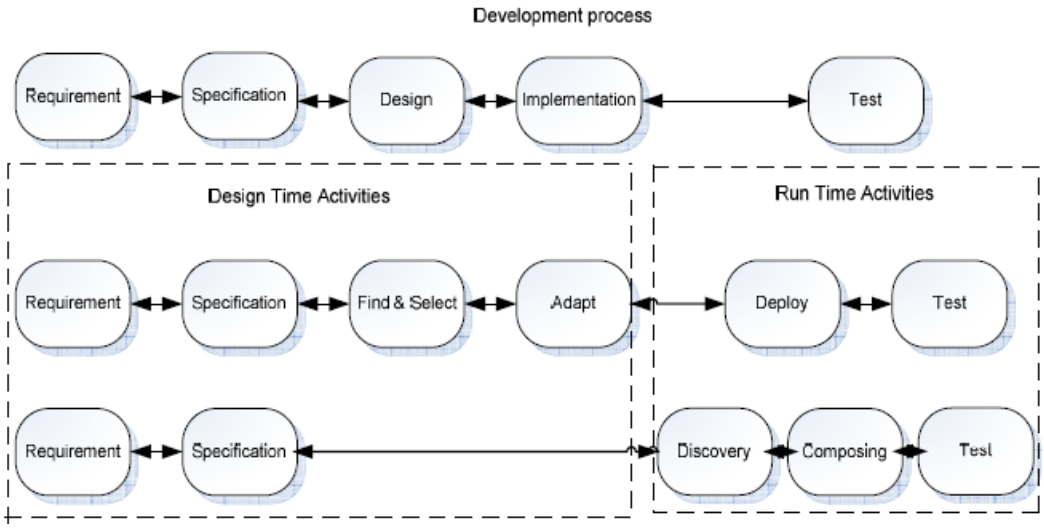


Figure 1.1. Comparison of typical activities during the development process in CBSE and SOSE. The top row lists the steps in the software engineering process. The middle row shows the step in CBSE, which is more focused on finding existing components, whereas the bottom row shows the steps in SOSE, which is driven more by identifying services as represented by the dashed box to the right (Breivold and Larsson 2007)

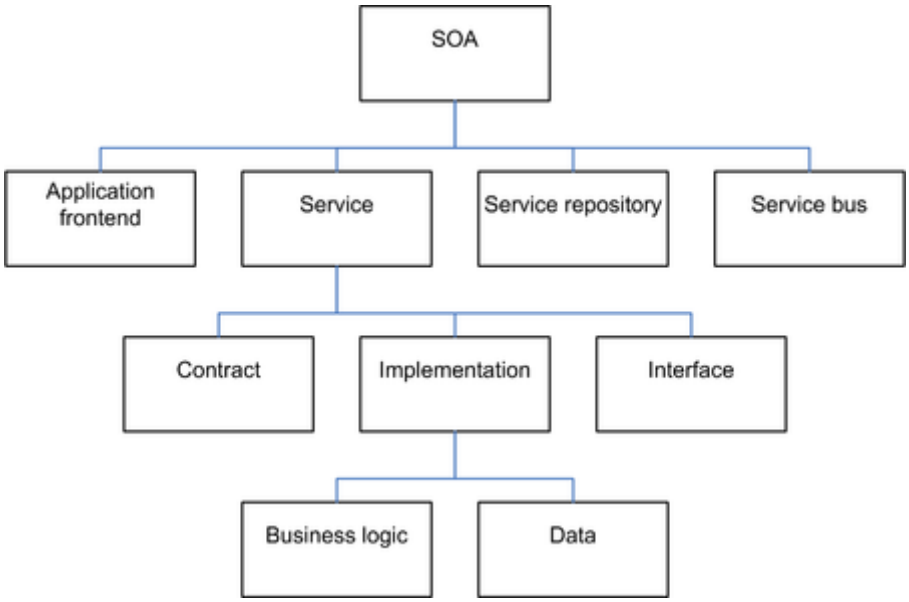


Figure 1.2. Elements of SOA (Dirk Krafzig, Karl Banke, and Dirk Slama, *Enterprise SOA*. Prentice Hall, 2005). The fundamental element is the Service, which is made fulfilled using contracts, implementations, and interfaces. Analogously, the service is available through the registry, a bus, and a frontend.

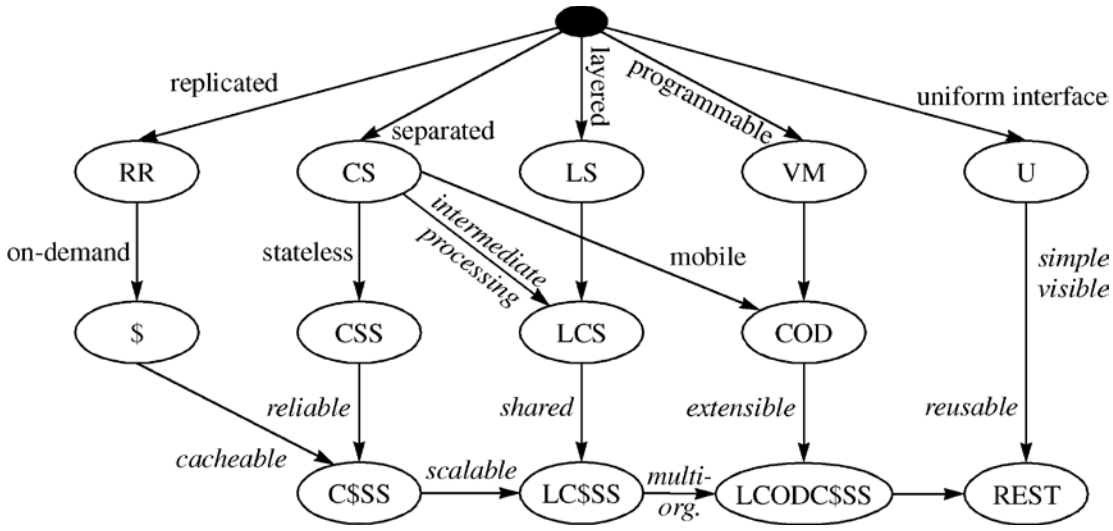


Figure 1.3. Schematic of styles contributing to REST (Representational State Transfer): RR = replicated repository; $ = cache; CS = client-server; CSS=client stateless server; C$SS = client cache stateless server; LS = layered system; S = stateless; VM = virtual machine; COD = code on demand; U = uniform interface (Fielding and Taylor 2002).

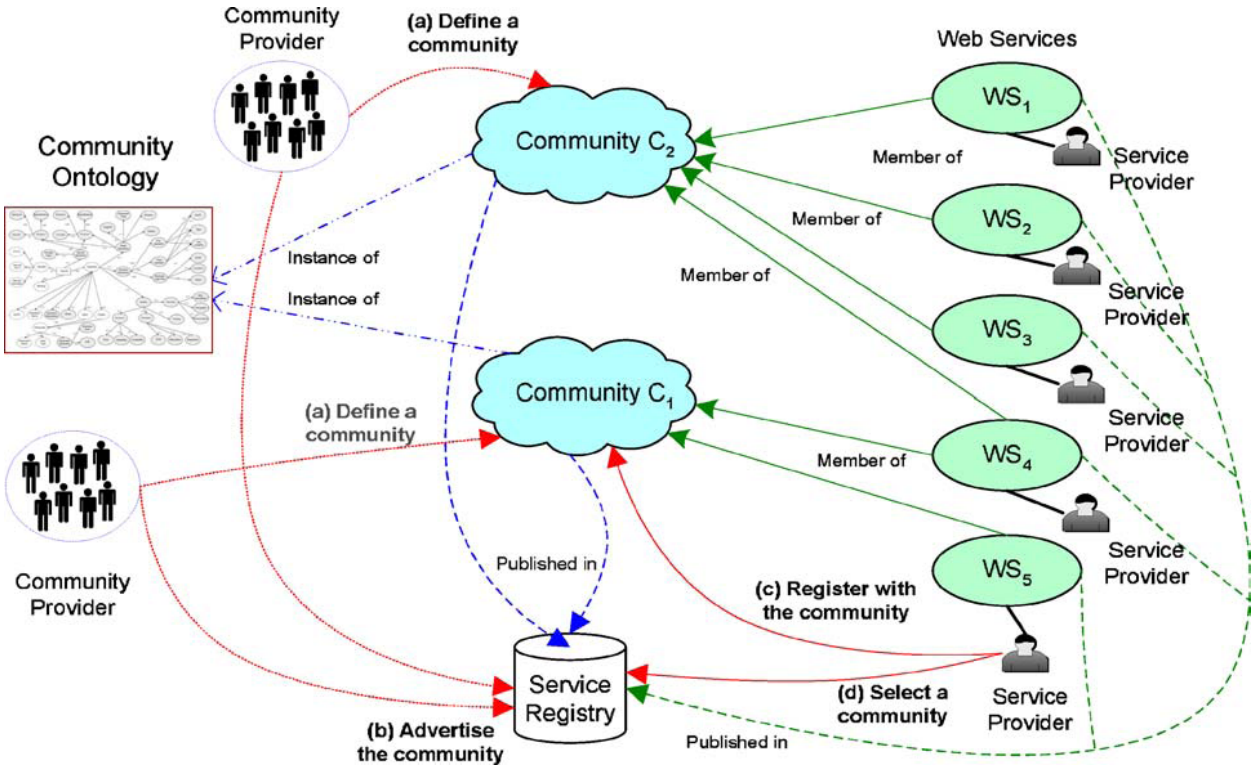


Figure 1.4. Model for Semantic Web Services (Medjahed and Bouguettaya 2005). (a) The community providers assign values to the attributes of the metadata ontology (community ontology), which (b) advertises its requirements with a service registry. (c) A service provider then identifies the community, of which it may be a member, and (d) registers its services with the community.

2. GoogleVision

Preliminary work has been completed on a component-based Web application using Google Earth to view user-supplied images of model results. The pictures were generated using MATLAB and stored on the local filesystem as part of previous studies. This work will be described first because is one of the fundamental components of a potential Web API for navy purposes.

Overview of Google Earth

Google Earth is a virtual globe, map and geographical information program that was originally called EarthViewer 3D, which was created by Keyhole, Inc, a company acquired by Google in 2004. It maps the earth's surface by the superpositioning of satellite and aerial images on a 3D globe. User data can also be added as layers. Google Earth is also a Web Map Service client. Three-dimensional geospatial data are implemented using the Keyhole Markup Language (KML).

The internal coordinate system of Google Earth is geographic coordinates (latitude/longitude) on the World Geodetic System of 1984 (WGS84) datum (Kumar 1988; True and Ieee 2004). A general perspective projection is used, which is similar to the orthographic projection, except that the point of perspective is near-earth rather than at infinity (deep space). Images' dates vary. The image data can be seen from squares made when *Digital Globe Coverage* is enabled. The date next to the copyright information is not the correct image date. Zooming in or out could change the date of the pictures. Many of the international urban image dates are from 2004 and have not been updated whereas most U.S. images are kept current. The baseline resolution is 15 m, but higher resolution (~ 1 m) is available for U.S. and Europe.

Google Earth incorporates a layer called *Geographic Web* that includes integration with Wikipedia and Panoramio (i.e. a mashup). The *Google Ocean* feature allows users to zoom below the surface of the ocean and view the 3D bathymetry; it supports ~ 20 content layers. *Historical Imagery* allows users to access earlier images of many locations.

Currently, every image created from Google Earth using satellite data provided by Google Earth is a copyrighted map. Any derivative from Google Earth is made from copyrighted data that, under United States copyright law, may not be used except under the licenses Google provides. Google allows non-commercial personal use of the images (e.g. on a personal website or blog) as long as copyrights and attributions are preserved. By contrast, images created with NASA's globe software, World Wind, use the Blue Marble, Landsat, or USGS layers, which are in the public domain and can be freely modified, redistributed and used for commercial purposes.

The Google Earth API is a free beta service, available for any web site that is free to consumers. The plug-in and its JavaScript API allow users to place a version of Google Earth into web pages. The API does not have all the features of the full Google Earth Application but enables 3D map applications. The Google Earth plug-in is currently available for Microsoft Windows (2000, XP, Vista, and 7) and Apple Mac OS X 10.4 and higher (Intel and PowerPC). To date the plug-in supports the following layers:

- Terrain

- Roads
- Buildings
- Borders
- 3-D Buildings

Data importing features and techniques

Google Earth supports several methods for importing and displaying user-supplied data. These were explored as potential pathways for entering both observations and model results. The free version only imports image files or text. Google Earth Pro can import a variety of formats, including vector data. These topics are explained in detail in the user guide (http://earth.google.com/userguide) and are only introduced here.

KML FILES

The Keyhole Markup Language is an XML grammar and file format for modeling and storing geographic features such as points, lines, images, polygons, and models for display in Google Earth, Google Maps and other applications. You can use KML to share places and information with other users of these applications. You can find example KML files on the KML Gallery and Google Earth Community site (http://bbs.keyhole.com). A KML file is processed by Google Earth in a similar way that HTML and XML files are processed by web browsers. Like HTML, KML has a tag-based structure with names and attributes used for specific display purposes. Thus, Google Earth acts as a browser of KML files (i.e., earth browser). You can use KML to:

- Specify icons and labels to identify locations on the planet surface
- Create different camera positions to define unique views for each of your features
- Use image overlays attached to the ground or screen
- Define styles to specify feature appearance
- Write HTML descriptions of features, including hyperlinks and embedded images
- Use folders for hierarchical grouping of features
- Dynamically fetch and update KML files from remote or local network locations
- Fetch KML data based on changes in the 3D viewer
- Display COLLADA textured 3D objects

IMAGE OVERLAYS

It is possible to put an image on top of Google Earth's surface in the 3D viewer. Furthermore, there is a way to make Super-Overlays (explained below). There are three types of specific overlays: PhotoOverlay, ScreenOverlay, and GroundOverlay. An overlay is created by specifying what image file to display (from computer disk, LAN, or WWW), how to fit its boundaries, and where it is located. Most image formats are accepted, except for EPS and PS files. The PNG and GIF formats can be modified to change the transparency by the viewer. The overlay image must have a North-Top orientation and be a simple cylindrical projection (i.e. Lat/Lon WGS84 projection). A PhotoOverlay allows for specific viewpoints on a certain sector of Earth in a rectangle, full or partial cylinder, or sphere. A ScreenOverlay is used to show static images like a logo, a compass, or HUDs (heads-up displays). A GroundOverlay drapes an

image on to the terrain. An overlay can be created interactively using a dialogue box but this method is not used in this work because it is too labor intensive for multiple images.

You can place Web Mapping Service (WMS) image overlays in Google Earth. These are mapping images that are provided through a WMS over the Internet. They can show you specific information, such as weather maps and charts, topographical maps, alternate high resolution satellite imagery.

NETWORK LINKS

This feature allows for KML or KMZ files on a network (LAN or WAN) to be saved on a specific webpage that can be linked to through the KML/KMZ file. This allows automatic updating of images into Google Earth. If utilized with Regions, Network Links have the possibility of loading large amounts of data. KML network links import static data as well as data streaming using a CGI (Common Gateway Interface) script or compiled program (e.g., PHP, C, Python, and Perl).

Some general guidelines for using network links are included in Table 2.1. A network link contains a <Link> element with an <href> (a hypertext reference) that loads a file. The <href> can be a local file specification or an absolute URL. Despite the name, a <NetworkLink> does not necessarily load files from the network. The <href> in a link specifies the location of an image file used by icons in icon styles, ground overlays, and screen overlays, a model file used in the <Model> element, and a KML or KMZ file loaded by a Network Link. The specified file can be either a local file or a file on a remote server. In their simplest form, network links are a useful way to split one large KML file into smaller, more manageable files on the same computer.

There are a few important rules to follow in creating network links. These are briefly stated here and explained using the Patos Lagoon example. The tutorial from Google Earth states that relative path is allowed, but this only applies for directories below the server directory. The browser and other Web applications cannot see the file system as such. It is acceptable to use links, however, as long as they are created on the server; short cuts do not work the same for Google Earth. One of the best reasons for using a network link is to implement dynamic web services. The content can be modified and served either by generating new KML files or by using links on the server side. This will be discussed more below.

REGIONS

Regions are a powerful KML feature that allows you to add very large datasets to Google Earth without sacrificing performance. Data is loaded and drawn only when it falls within the user's view and occupies a certain portion of the screen. Using Regions, you can supply separate levels of detail for the data, so that fine details are loaded only when the data fills a portion of the screen that is large enough for the details to be visible. Any Feature can contain a Region. A Region affects visibility of a Placemark's geometry or an Overlay's image. Regions define both culling and level-of-detail behavior of the affected geometry or overlay. Regions are inherited through the KML hierarchy and affect the visibility of Features that are defined lower in the hierarchy.

In KML, some classes are derived from a base class. For convenience, we refer to the base class instead of listing all of the derived classes. For example, *Feature* refers to any KML element that is derived from Feature (Document, Folder, GroundOverlay, NetworkLink, PhotoOverlay, Placemark, and ScreenOverlay). *Geometry* refers to any geometric element in KML (i.e., Point, Polygon, LinearRing, LineString, Model, and MultiGeometry). *Overlay* refers to the elements derived from Overlay: GroundOverlay, PhotoOverlay, and Screen Overlay.

A Region has a <LatLonAltBox> that defines a bounding box for your data. Objects associated with this bounding box are drawn when the Region comes within view and the projected onscreen size of the <LatLonAltBox> falls within the specified pixel range for that Region. The second important concept related to Regions is that of *Level of Detail*, or LOD for short. Since a computer screen has a limited amount of space, it is most efficient to set things up so that large amounts of data are loaded only when there are enough pixels to display the data adequately. When the Region is taking up a relatively small portion of the screen, the LOD mechanism allows the KML author to specify a dataset with lower resolution to be substituted for the full-resolution data. This lower-resolution dataset loads faster. You can also specify a *fade extent* for a Region, which allows an object to transition gracefully from transparent to opaque, and back again. Fade ranges are optional, but they prevent the "pop" effect between LineStrings or Polygons of different resolutions. Fading is very expensive in terms of performance and should not be used with imagery.

A common use of Regions is to nest them with the larger Regions associated with coarser resolution and the smaller, inner Regions associated with increasingly finer LOD. As the user's viewpoint moves closer, Regions with finer LOD replace the previously loaded Regions with coarser LODs.

SUPER-OVERLAYS

A super-overlay is a hierarchy of Regions and NetworkLinks that can be used to serve a large set of imagery efficiently. Tiles of appropriate resolution are loaded as portions of the imagery area come into view, with higher resolution tiles loaded as the viewpoint nears. It's a waste of effort to try to display a 7008-by-6720-pixel image on a 1024-by-768 screen. What's more, if the user is miles above the earth's surface, all that data might be crammed into a handful of pixels, and performance is dismal. Super-overlays allow you to take advantage of NetworkLinks and their ability to load the data associated with a Region if it is active (i.e., within view and whether its projected size is appropriate to the current viewpoint). If you subdivide the original image into a hierarchy of images with increasing levels of detail, Google Earth can load the imagery that best fits the current view.

To produce a super-overlay, an original (large) image is subdivided into hundreds of small GroundOverlays. These overlays, or *tiles*, (Sample, 2010; Potmesil 1997) are arranged in a hierarchy. To create a super-overlay, you need to: (1) Prepare the imagery by subdividing it into manageable chunks (256-by-256 pixels is recommended), and (2) Create the KML files that set up the Regions, Links, Network Links, and, in this case, the files containing the GroundOverlays. For each image, prepare a KML file that associates the ground overlay with a Region and a NetworkLink. Each KML file in this set has the following elements:

- Region (with LatLonAltBox, minLodPixels, and maxLodPixels so that Google Earth can determine whether the Region is active at any given time)
- set of NetworkLinks to the child files (the tiles in the next level of the hierarchy)
- the ground overlay for this Region

COMMON GATEWAY INTEFACE (CGI)

CGI is the standard that defines how the Web server software delegates the generation of web pages to a text-based application (Gundavaram, 1996). These are commonly known as CGI scripts, which can be written in any programming language, but scripting is often used. The CGI provides an interface between the Web server and the client, usually a Web browser. The creation of CGI scripts requires a Web server. In the future, a Web server can and should be used to create and update KML codes.

PLACEMARKS

Placemarks are one of the easiest features to put into Google Earth. They can be created using the Placemark tool. Name the placemark, and in the description box a link, image, chart, or even a video can be embedded into the description. Place WGS84 coordinates in the appropriate text boxes. It is more convenient for automation to use a KML file to determine placemarks. You can specify a custom icon for the Placemark, and add other geometry elements to it. There are three kinds of place marks: *simple*, *floating*, and *extruded*. The KML code for a simple placemark (Figure 2.1) includes an XML header (line 1 in every KML file), a KML namespace declaration (line 2 in every KML 2.2 file), and a <Placemark> object. The placemark contains the following elements: a <name> that is used as the label; a <description> that appears in the balloon attached to the Placemark; and a <Point> that specifies the <coordinates> of the Placemark on the Earth's surface (*longitude, latitude,* and optional *altitude*).

What users commonly think of as a placemark in Google Earth is actually a <Placemark> element with a <Point> child in KML. A Point Placemark is the only way to draw an icon and label in the 3D Viewer of Google Earth. By default, the icon is the familiar yellow pushpin. In KML, a <Placemark> can contain one or more geometry elements, such as a LineString, Polygon, or Model. But only a <Placemark> with a Point can have an icon and label. The Point is used to place the icon, but there is no graphical representation of the Point itself. You can add links, font sizes, styles, and colors, and specify text alignment and tables (see http://code.google.com/apis/kml/documentation/KML_Samples.kml).

Animation notes

Google Earth does not allow animations (e.g., animated GIF) to be used as overlays. To animate in Google Earth, a series of overlay image files are imported with time data included in a KML file. However, you could embed animations into the <description> popup in the tags, as well as embedding videos in the description. The method used in this work (Figure 2.2) is discussed in the Atchafalaya Bay example.

The <TimeSpan> tag encloses <begin> and <end> tags, which indicate the start and end dates for each image within a loop. Date and time are defined using XML schema time: yyyy-mm-

ddThh:mm:ss, where hours are 0 to 23. The time stamp was computed uniquely for the examples discussed in this report because of time inconsistencies in the original files. This problem can be avoided by following the XML schema time for file names. It also works in ArcGIS Explorer™; however, a colon ":" cannot be part of a file name for Windows platforms.

When importing an image into Google Earth using WGS84 coordinates, it may be necessary to rotate and resize the image to make it display properly in the 3D viewer. This requires some user action and several adjusmetnts of the coordinates in the image's properties's location tab (or the KML file). It is important to convert the coordinates to decimal form because KML does not parse minutes and seconds.

Creating KML files with Perl scripts

It is possible to use Perl scripts to generate KML files that would otherwise be very tedious to create. The basic block within the KML file that needs to be reproduced for each time-stamped image file to be displayed consists of only a few tags (Figure 2.2). The first two lines are required for the KML parser. The 3D viewer will import the images into a folder as indicated by the <Folder> tag (e.g., Atchafalaya_Tracer). The individual image will be a <GroundOverlay> named "tracer_1997-12-21T00:00:00.gif". The <TimeSpan> tag indicates the interval, which is taken as the time interval for which this image is valid. For this example, the <begin> date is the date of the file. The <Icon> tag is a Link-Object, which holds the Web address of the file to be imported. The last set of tags defines the <LatLonBox> to use as the boundaries of the image. The last two lines close the </Folder> and </kml> tags. They come after all desired files are inserted. If only one image were to be imported, this could all be done by hand, but the large number of files typically used (eight for this example) requires automated processing.

We have created KML files using Perl scripts during the GoogleVision project. These scripts are very convenient for writing out text files because of the ease of processing strings and numbers with Perl. An example script (Figure 2.3) shows the components of the KML creation process. A listing of required image files to import into Google Earth is obtained using the operating system as shown at statement (1). The KML file is opened and the header and main tags inserted at (2). Assign the bounds by trial and error at (3). This is uncertain because the images may include white space around the plot and the Lat/Lon must be found for the edges of this white space. We ran the Perl script and opened the resulting KML file in the 3D viewer to adjust these coordinates. We loop over all of the files from the specified directory at statement (4) and replace the file name with the XML schema discussed above. Write the KML body out with the Perl variables $start_dtg and $end_dtg in the <TimeSpan> tag. Place the other variables in the appropriate tags and write the footer to the file after the loop completes. An example of the image overlay can be seen in Figure 2.4. Note that the image includes the entire page, whose coordinates had to be placed in the <LatLonBox> tag.

Examples

This section will present results using the methods developed under the GoogleVision project. Each of the sample regions incorporates some of the techniques described above. The original model simulations were completed during several research projects at NRL (Cobb et al. 2008; Keen and Holland 2010). They are shown here to demonstrate the GoogleVision methods.

If a Windows OS computer is used to construct the images and place them in the web folders, the permissions must be changed manually on the Linux computer that is hosting the web page to allow reading by the world (i.e. chmod 744). The default permissions are 740, which do not allow reading by the web server. Another very important requirement is that the latitiude and longitude for ALL images referenced in a KML file to be opened by GoogleEarth be in the same reference system (e.g., longitudes all negative or all positive; i.e., no mixing as in -90 and 270).

ATCHAFALAYA BAY, LOUISIANA

This example demonstrates several useful techniques: (A) displaying a series of image files as an animation using a Perl-generated KML file; (B) a placemark; (C) a screen overlay; and (D) a balloon with user-supplied data. No local files are used. It also demonstrates two ways to introduce user-supplied data—as screen overlays and links that can be opened in a browser. All of these displays can be shown on the same view (Figure 2.4).

The KML file (Figure 2.2) for the initial image file of the simulated tracer distribution (Figure 2.4A) contains only a few tags to display one image. This file produces a folder named "Atchafalaya_Tracer" in Google Earth; within this folder is a GroundOverlay file named "tracer_1997-12-21T00:00:00.gif", which is applicable for time span <begin> to </end>. An icon (image) is produced from the link, http://gserver/Atchafalaya/CONC1_z1m_h000.gif, with corners defined by the <north>, etc. tags. The Perl script that produces the full kml file for all images is shown in Figure 2.3.

The Google 3D viewer also shows a placemark at Cypremort Point, as indicated by the standard thumbtack icon (Figure 2.4B). This icon can be changed by the user. In order to display the icon, the <LookAt> tag is required (Figure 2.5). The placemark location is set by the coordinates of the point using the tags. Note that the longitude needs to be entered twice for the placemark to be displayed and the <range> is the distance in meters to the target from the viewer (altitude). Additional data can be displayed in the 3D viewer using a screen overlay (Figure 2.4), in the example this is the timeseries plots of wind to the left. The screen overlay does not move within the viewer but remains fixed. The example is placed in its current position using a KML file (Figure 2.6) that defines the overlay and screen reference points using <overlayXY> and <screenXY> tags, respectively, where the XY tags range from 0 to 1. Note that the file is located on the server. This file was created in a text editor but it could easily be created for dynamic data using a script or program.

A balloon feature can be used to insert a variety of user-supplied data. The balloon used in the Atchafalaya project (Figure 2.4D) is colocated with the Cypremort Point placemark. When clicked, its icon (the default is the same as for a placemark) opens a balloon containing links to the same images that are displayed in the screen overlay. The KML file (Figure 2.7) that was used shows only one way the balloon feature can be used. This example defines a balloon template with the <Style> and <BalloonStyle> tags. A placemark is then identified and the <ExtendedData> tag is used to incorporate the selected images. Note that the placemark using the "dataBalloon" style defined in the file, where "data1" and "data2" are dummies for the image locations. As with the simple placemark, the <LookAt> tag is required to display the balloon. The click event is a part of the extended data. Also, note the <range> tag, which defines the distance in meters from the placemark for displaying the point.

PATOS LAGOON, BRAZIL

The model results for the Patos Lagoon simulations are displayed in the Google Earth 3D viewer (Figure 2.8) in a similar manner as Atchafalaya Bay. The animation consists of a series of images (Figure 2.8A). The additional feature (Figure 2.8B) is a dynamic screen overlay that will plot whatever new image is available. This section discusses this method.

The main web page for Patos Lagoon is “Cassino_Beach_Main.htm”. The table for links has an <href> line with a reference to a KML file called “Todays_Image.kml” (Figure 2.9), which is the static part of the Web service. The main tag in this KML file is the <NetworkLink> tag, which has a name and description in addition to a <Link> tag for the actual target. Note that the target is a KML file. This file would be generated by a script. However, the link in “Todays_CrossSection.kml” (Figure 2.10) refers to a static image file. This file, “Todays_CrossSection.gif” is a link created on the server to a file with a more specific name. This operation could have been accomplished as easily using a script to create the KML file that referred to the original file rather than a link.

ST. LOUIS BAY, MISSISSIPPI

This example shows how Regions can be used to display different levels of detail in a simplified manner. The basic functionality that makes it possible for GoogeEarth to display the range of data in such an efficient manner is the Region. There are at least two ways this construction can be used to display images. A very useful method is to use regions to display more detail for the same area. This concept is shown for the model bathymetry for St. Louis Bay in Figure 2.11. The axes are omitted because they were not useful for the overlapping images of this example. A more useful example, which also shows how the images overlap on the earth layer, demonstrates the increased resolution for vectors, which are always difficult to examine (Figure 2.12). The vectors smoothly increase in density as the view zooms in or pans around the domain. The highest resolution for this simulation was ~100 m. Note the coarser resolution vectors to the right (the next quadrant was not processed for this example).

A second technique is to plot inner nests within a larger domain. These higher resolution grids can be displayed as the view moves over them. Figure 2.13 demonstrates the result for the Mississippi Bight/Sound and St. Louis Bay area. The script (Figure 2.14) shows how <Folder> is used to nest subregions. The previous example (Figure 2.12) used a <Folder> within the intermediate plot to introduce the highest resolution images. The nested method is much easier to complete and produces a smoother result but it is not always applicable. These samples were completed manually but it would be necessary to use a script of some kind, and probably a better graphics program, to make this a powerful data visualization tool.

Another feature becomes available when multiple Regions are used. The transparency of the different layers can be adjusted to permit overlapping image viewing. This can allow, for example, vectors to be plotted over the model grid in GoogleEarth without creating special images. This is demonstrated in Figure 2.15.

SAN FRANCISCO BAY AND HUNTERS POINT

The San Francisco Bay example is very similar to the St. Louis Bay case except that the smallest model grid uses a resolution of only 30 m. The scale change from the full bay model (Upper panel in Figure 2.16) to the Hunters Point grid (lower right panel) is substantial. These

images were constructed directly from MATLAB without subsectioning as in the St. Louis Bay. One consequence of this is that the highest resolution data cannot be presented. Compare to Figure 2.12C to see the difference in resolution. In order to achieve this, it will be necessary to find a more automated tool for making the plots. Some adjustment of the <Lod> controls (Figure 2.14) is necessary for a smooth transition between Regions as well.

```
<?xml                    version="1.0"                    encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple                                              placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at    the    height    of    the    underlying    terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Figure 2.1. Simple placemark KML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>Atchafalaya_Tracer</name>
            <GroundOverlay>
            <name>tracer_1997-12-21T00:00:00.gif</name>
            <TimeSpan>
                  <begin>1997-12-21T00:00:00</begin>
                  <end>1997-12-21T03:00:00</end>
            </TimeSpan>
            <Icon>
                  <href>http://gserver/Atchafalaya/CONC1_z1m_h000.gif</href>
            </Icon>
            <LatLonBox>
                  <north>30.080356</north>
                  <south>28.658767</south>
                  <east>-90.798731</east>
                  <west>-92.784731</west>
            </LatLonBox>
            </GroundOverlay>
</Folder>
      </kml>
```

Figure 2.2. Example kml file to display one image file at a time. This block would be repeated for other time-dated images through the <TimeSpan> tags.

```
#!/usr/bin/perl
use POSIX;
# use the original file names for WinOS !!
#
$filedir = "Atchafalaya";
@files = `cd $filedir ; ls | grep gif`;                                    (1)

open FILE, ">Atchafalaya_Tracer.kml" ;                                     (2)

my @kmlheader = qq(<?xml version="1.0" encoding="UTF-8"?>
 <kml xmlns="http://www.opengis.net/kml/2.2"> ) ;
print FILE "@kmlheader" ;
my @kmlheader = qq(
  <Folder>
    <name>Atchafalaya_Tracer</name> );
print FILE "@kmlheader" ;

$number = $count ;
my $count = ("0") ;
# get the dates from the renamed files--these don't seem to work with
WinOS!!
#
$dir = "$filedir/STD_DTG_FILES";
@filedates = `cd $dir ; ls | grep gif`;

$east = -90.798731;                                                        (3)
$west = -92.784731;
$north = 30.080356;
$south = 28.658767;

$start_dtg = "1997-12-21T00:00:00";
$end_dtg = "1998-01-18T21:00:00";
$timestep = 3;
#print @files;
$nn = 0;
$nt = 0;
$maxn = scalar(@files);
print"$maxn\n";

foreach $file ( @filedates) {                                              (4)
      $infile = $files[$nn];
      $file1 = $filedates[$nn];
      chomp ($file1);
      chomp ($infile);
      $start = index ($file1, "_") + 1;
      $stop  = index ($file1, ".");
      $cnt = $stop - $start;
      $start_dtg = substr($file1, $start, $cnt);
      if ($nn < $maxn-1 ) {
            $file2 = $filedates[$nn + 1];
            chomp ($file2);
            $end_dtg = substr($file2, $start, $cnt);
      } else {
            $file2 = $file1;
            chomp ($file2);
```

```
            $start = index ($start_dtg, "T") + 1;
            $newdtg = substr($start_dtg, $start, 2);
            $end_dtg = $start_dtg;
            $cnt = 2;
            $hour += ($timestep * $nt);
            if ($hour > 21) { $hour = 0; }
            $newdtg       =       substr($end_dtg,       $start,       $cnt,
sprintf("%2.2d",$hour));
            if ( $nt > 7) {
                  $nt = 0;
                  $newday = 19;
                  $start = index ($start_dtg, "T") - 2;
                  $newdtg = substr($start_dtg, $start, 2);
                  $newdtg = substr($end_dtg, $start, $cnt,
            }
      }
      $nt++;
      $nn++;
      my @kml = qq(
            <GroundOverlay>
            <name>$file1</name>
            <TimeSpan>
                  <begin>$start_dtg</begin>
                  <end>$end_dtg</end>
            </TimeSpan>
            <Icon>
                  <href>http://gserver/$filedir/$infile</href>
            </Icon>
            <LatLonBox>
                  <north>$north</north>
                  <south>$south</south>
                  <east>$east</east>
                  <west>$west</west>
            </LatLonBox>
            </GroundOverlay>) ;
      print FILE "@kml" ;
}

my @kmlfooter = qq(  </Folder> );
print FILE "@kmlfooter" ;
my @kmlfooter = qq(  </kml>);

print FILE "@kmlfooter" ;
```

Figure 2.3. Example of a Perl script to create a kml file for a series of image files as ground overlays.

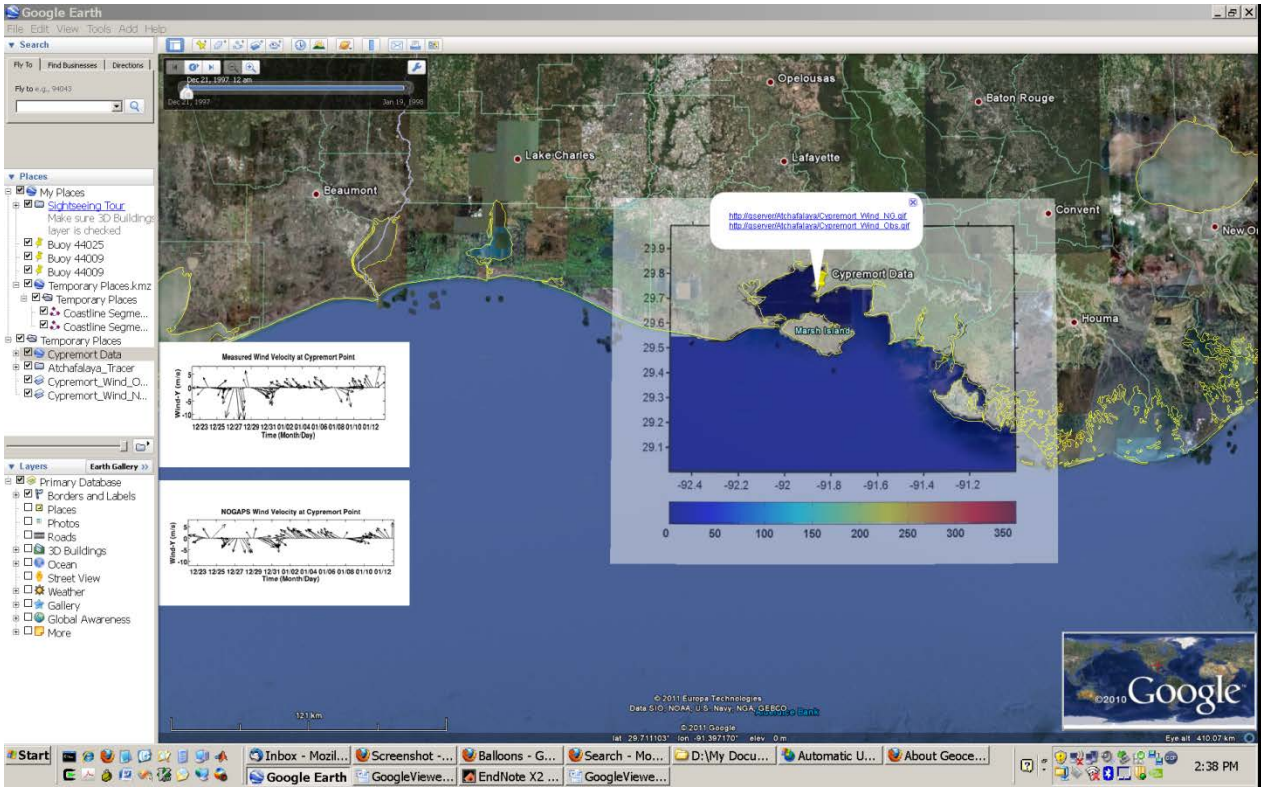


Figure 2.4. Screen shot of Google Earth showing the techniques discussed for the Atchafalaya example: (A) ground overlay of an image file; (B) placemark; (C) screen overlay; and (D) balloon. The ground overlay would be produced by the kml file in Figure 2.2 and animated using the Perl script from Figure 2.3 to create a kml file to overlay a series of time-stamped images.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">

      <Placemark id="cypremortPointWind">
            <name>Cypremort Wind Observations</name>
            <LookAt>
                  <longitude>-91.872</longitude>
                  <latitude>29.719</latitude>
                  <range>200440.8</range>
            </LookAt>

            <Point>
                  <coordinates>-91.872,29.719,0</coordinates>
            </Point>
      </Placemark>
</kml>
```

Figure 2.5. Example Kml file for a placemark icon in Figure 2.4B.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">

      <ScreenOverlay>
            <Icon>

      <href>http://gserver/Atchafalaya/Cypremort_Wind_NG.gif</href>
            </Icon>
            <overlayXY x="0" y="0" xunits="fraction" yunits="fraction"/>
            <screenXY x="0" y=".2" xunits="fraction" yunits="fraction"/>
      </ScreenOverlay>
</kml>
```

Figure 2.6. Sample Kml file for the simple screen overlay plotted in Figure 2.4C.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">

      <Document>
            <name>Cypremort Data</name>
            <Style id="dataBalloon">
                  <BalloonStyle>
                        <text>
                              <![CDATA[
                              $[data1]
                              $[data2]
                              ]]>
                        </text>
                  </BalloonStyle>
            </Style>
            <Placemark id="cypremortPointWind">
                  <name>Cypremort Data</name>
                  <styleUrl>#dataBalloon</styleUrl>
                  <ExtendedData>
                        <Data name="data1">

      <value>http://gserver/Atchafalaya/Cypremort_Wind_NG.gif</value>
                        </Data>
                        <Data name="data2">

      <value>http://gserver/Atchafalaya/Cypremort_Wind_Obs.gif</value>
                        </Data>
                  </ExtendedData>
                  <LookAt>
                        <longitude>-91.872</longitude>
                        <latitude>29.719</latitude>
                        <range>200440.8</range>
                  </LookAt>
                  <Point>
                        <coordinates>-91.872,29.719,0</coordinates>
                  </Point>
            </Placemark>
      </Document>
</kml>
```

Figure 2.7. Sample Kml file to display the balloon feature for the winds at Cypremort Point (Figure 2.4D).

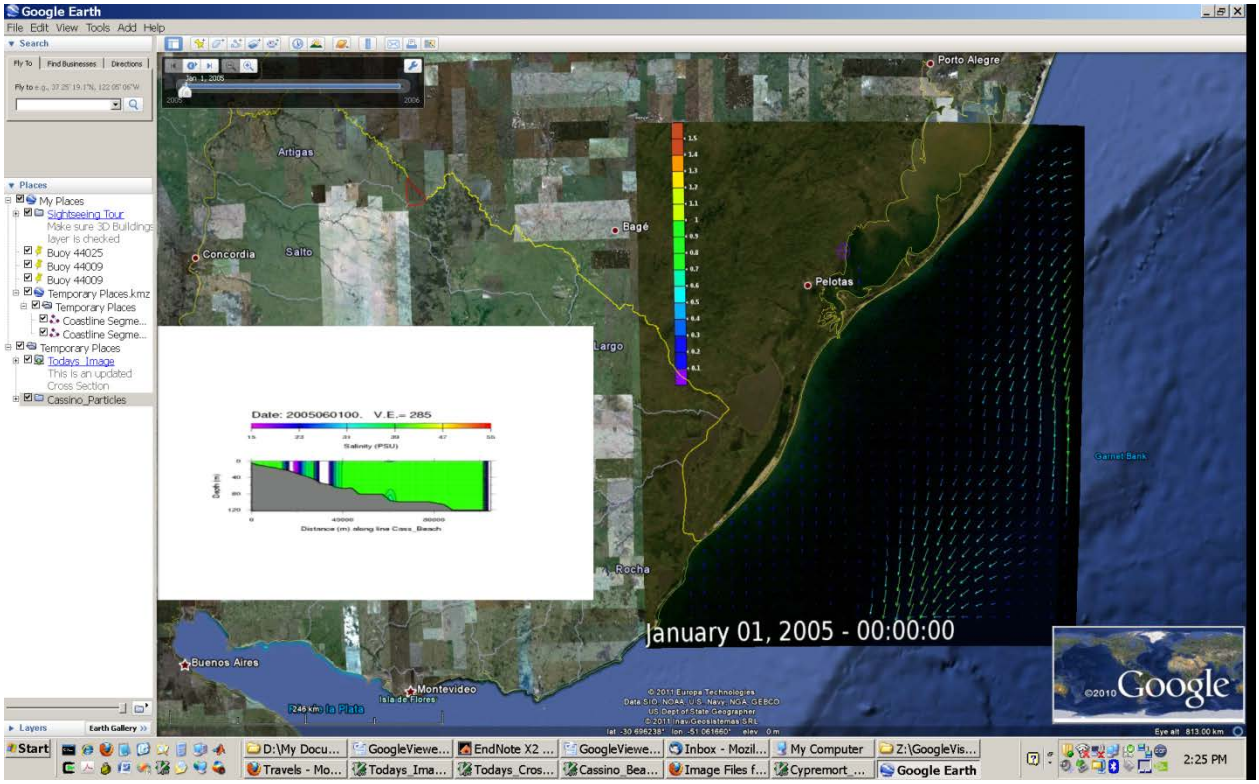


Figure 2.8. Screen shot of the Patos Lagoon data, showing (A) the particle simulations and (B) the network link to an updated image file for a cross-section of salinity.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">

      <NetworkLink>
            <name>Todays_Image</name>
            <visibility>0</visibility>
            <description>This is an updated Cross Section</description>
            <flyToView>1</flyToView>
            <Link id="txs">

      <href>http://gserver/Cassino/Todays_CrossSection.kml</href>
                  <viewRefreshMode>onRequest</viewRefreshMode>
            </Link>
      </NetworkLink>
</kml>
```

Figure 2.9. Example kml file to access a dynamically updated image file as in Figure 2.8A.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">

      <ScreenOverlay>
            <Icon>

      <href>http://gserver/RESULTS/Todays_CrossSection.gif</href>
            </Icon>
            <overlayXY x="0" y="0" xunits="fraction" yunits="fraction"/>
            <screenXY x="0" y=".2" xunits="fraction" yunits="fraction"/>
            <size x=".4" y=".4" xunits="fraction" yunits="fraction"/>
      </ScreenOverlay>
</kml>
```

Figure 2.10. Example Kml file (Todays_CrossSection.kml) used to load dynamic image files in Figure 2.8B.

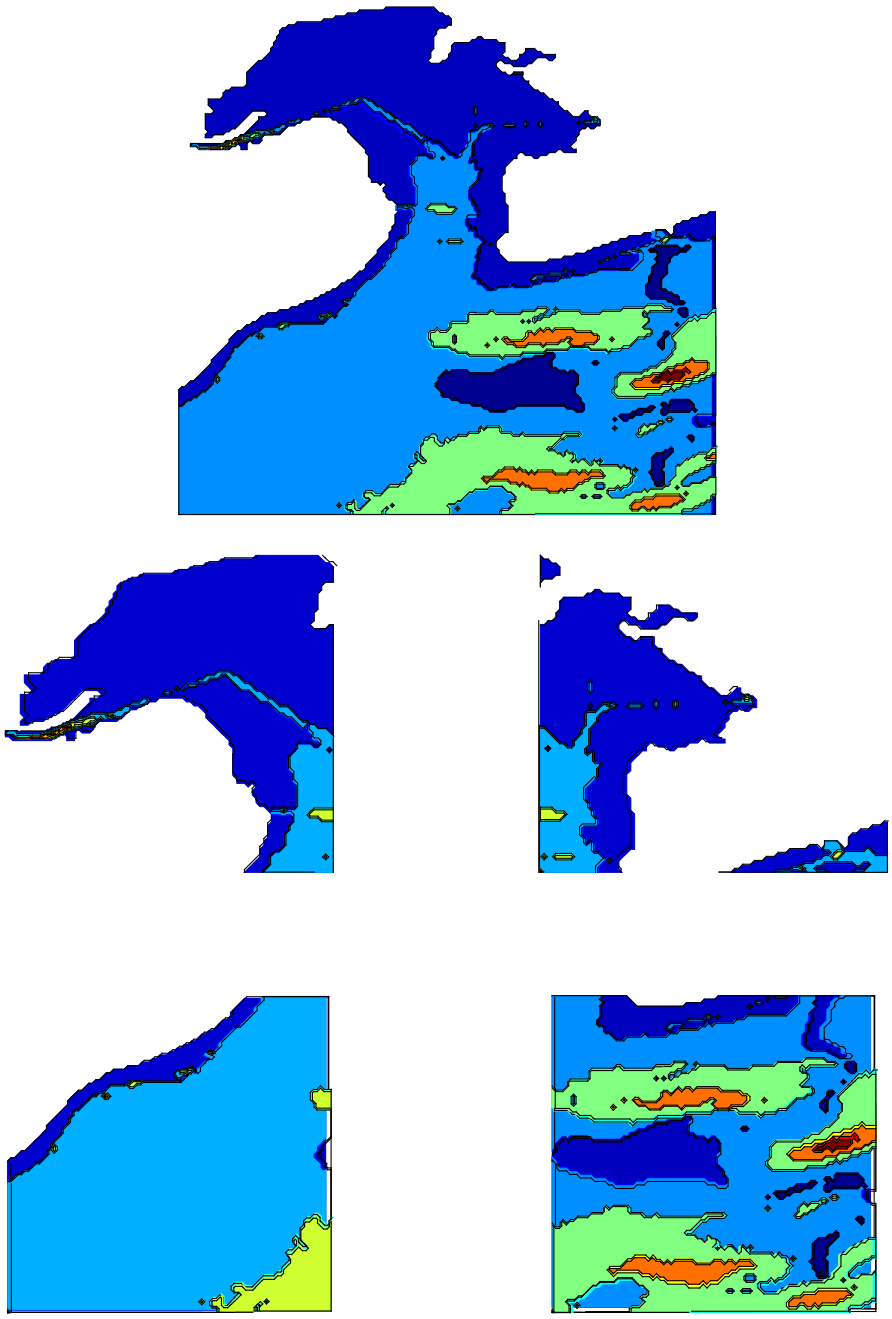
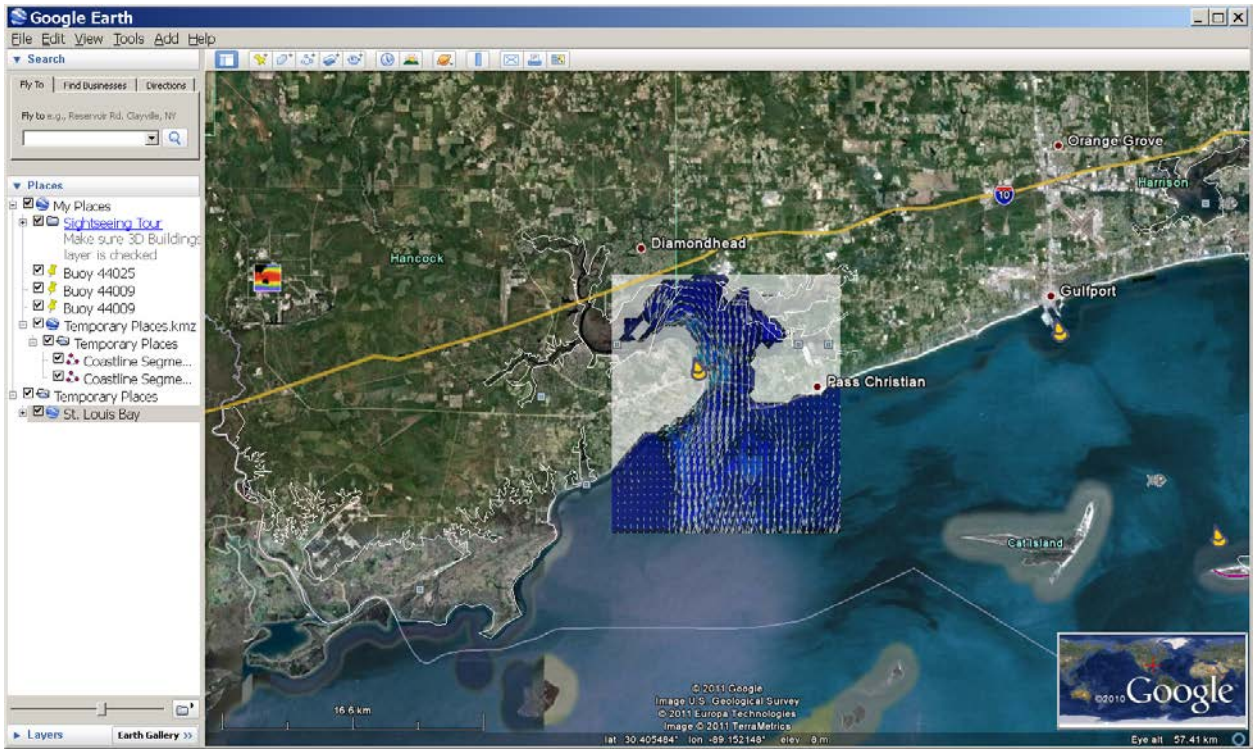
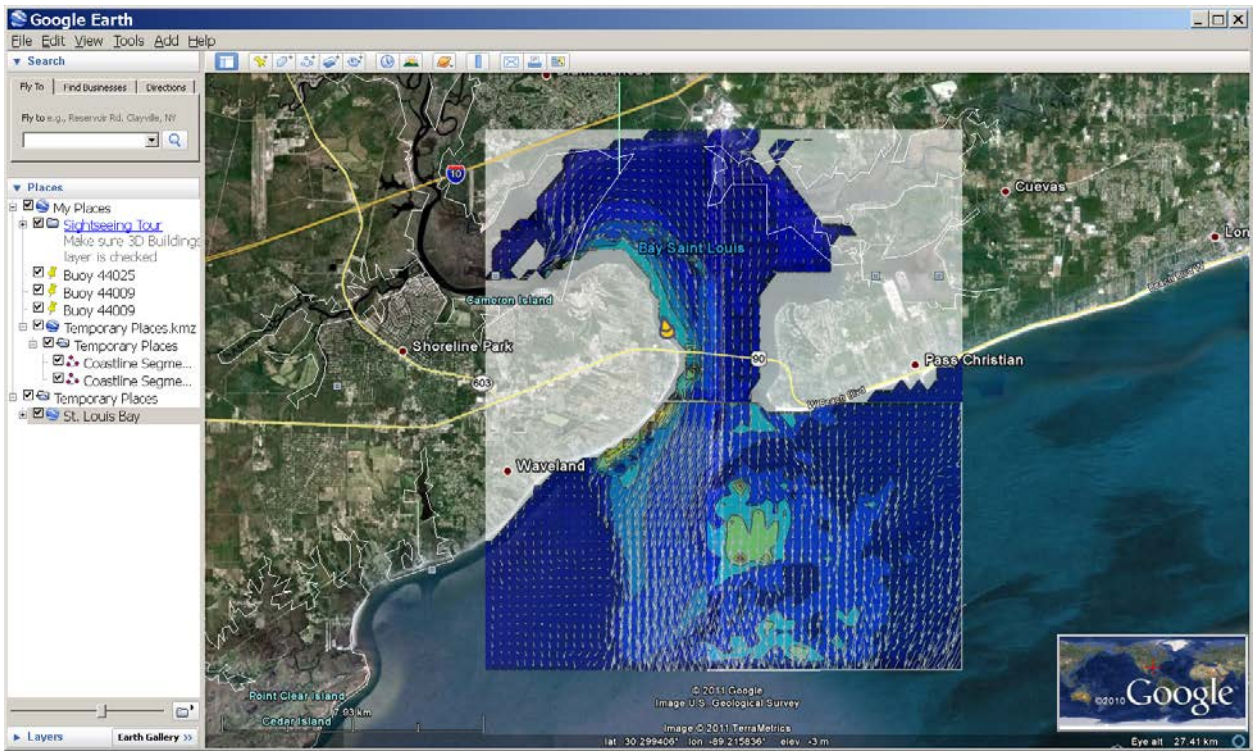


Figure 2.11. The top panel shows the St. Louis Bay model grid. The lower consists of the four sub-regions used to display higher resolution data. This plot uses the same number of points for all of the regions.


Google Earth
File Edit View Tools Add Help
Search
Fly To
Find Businesses
Directions
Places
My Places
Sightseeing Tour
Buoy 44025
Buoy 44009
Buoy 44009
Temporary Places.kmz
Temporary Places
Temporary Places
St. Louis Bay
Layers
Earth Gallery
Orange Grove
Harrison
Diamondhead
Hancock
Gulfport
Pass Christian
Cat Island
16.6 km
© 2011 Google
Image U.S. Geological Survey
© 2011 Europa Technologies
Image © 2011 TerraMetrics
Google
Eye alt 57.41 km


Google Earth
File Edit View Tools Add Help
Search
Fly To
Find Businesses
Directions
Places
My Places
Sightseeing Tour
Buoy 44025
Buoy 44009
Buoy 44009
Temporary Places.kmz
Temporary Places
Temporary Places
St. Louis Bay
Layers
Earth Gallery
Cuevas
Bay Saint Louis
Cameron Island
Shoreline Park
Pass Christian
Waveland
Point Clear Island
Cedar Island
7.93 km
© 2011 Google
Image U.S. Geological Survey
Image © 2011 TerraMetrics
Google
Eye alt 27.41 km

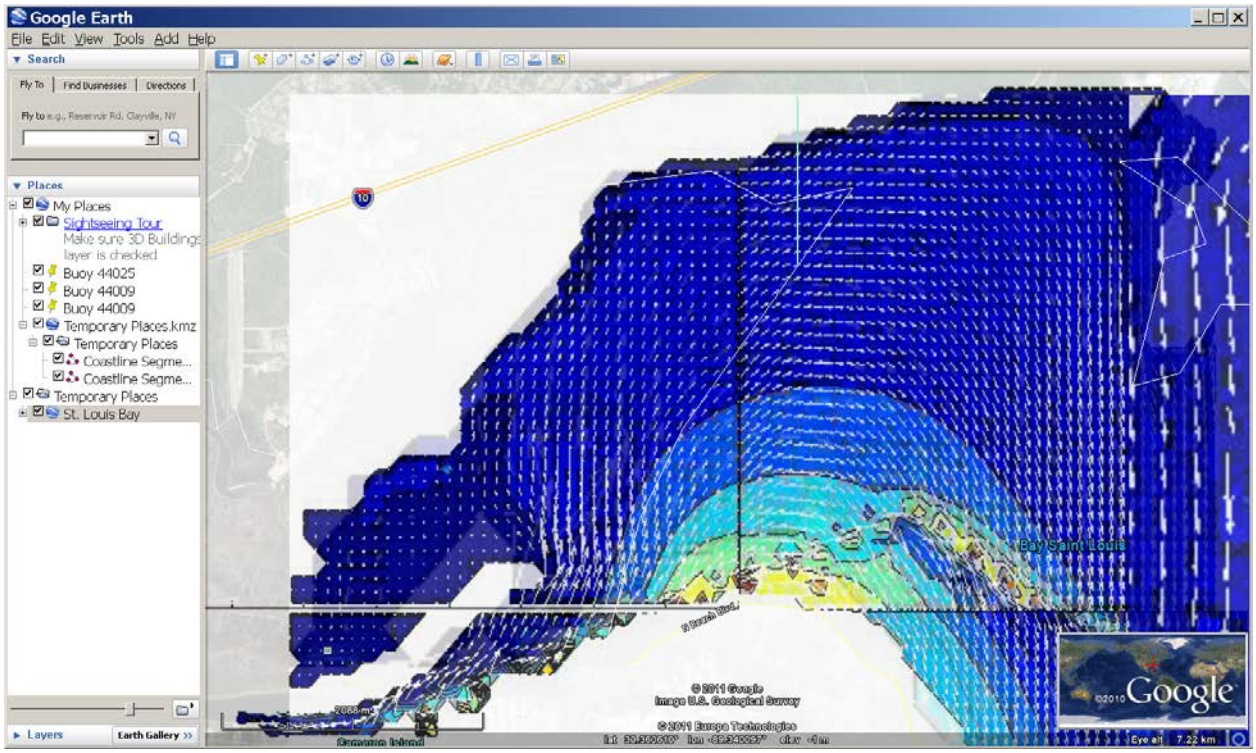


Figure 2.12. Regions plots of current vectors over contours of surface concentration of the tracer for the St. Louis Bay model. (Top) A plot of every 5th grid point. (Middle) Every 3 grid cell. (Bottom) Every grid cell. The <Regions> views are adjusted to minimize overlap while allowing a smooth transition between image sets. The maximum current is 1.6 m/s and the maximum concentration is 430 dimensionless units.

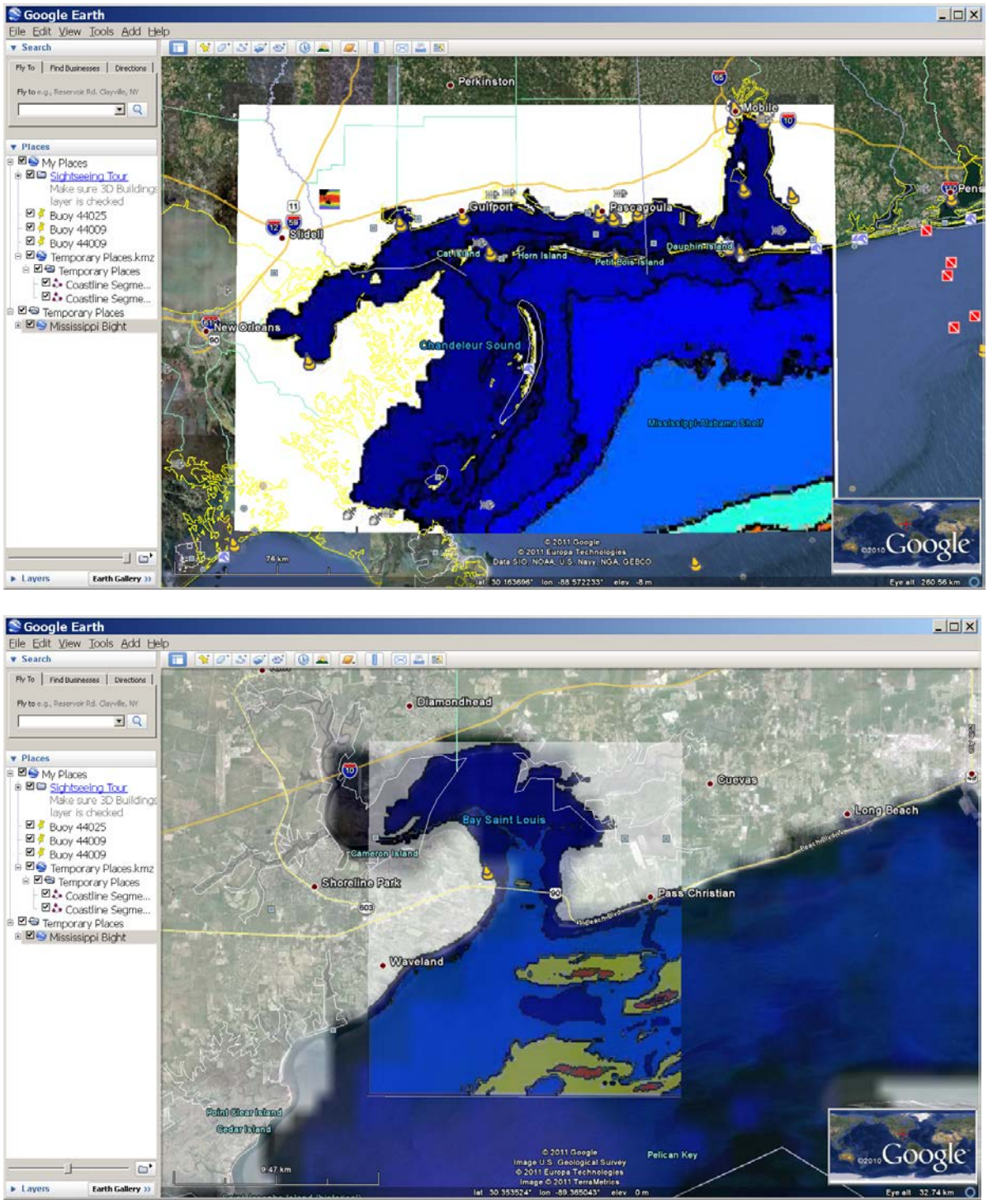


Figure 2.13. Images showing the use of Regions for nests. The top image is a screen print of the ~800 m Mississippi Bight grid. The lower image is the zoomed-in screen showing the 100 m grid with the coarse grid in the background. The script to set these regions up is shown in Figure 2.14.

```
<?xml gversion="1.0" encodin="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Mississippi Bight</name>
    <Region>
      <LatLonAltBox>
        <north>30.7166</north>
        <south>29.3048</south>
        <east>-87.66235</east>
        <west>-89.9552</west>
      </LatLonAltBox>
      <Lod>
      <minLodPixels>128</minLodPixels>
      <maxLodPixels>1024</maxLodPixels>
      </Lod>
    </Region>
    <GroundOverlay>
      <name>Mississippi Sound</name>
      <Icon>
        <href>SuperOverlay/MSB_BATHY_G1.jpg</href>
      </Icon>
      <LatLonBox>
……
      </LatLonBox>
    </GroundOverlay>
    <Folder>
    <Region>
      <LatLonAltBox>
        <north>30.38</north>
        <south>30.2325</south>
        <east>-89.2325</east>
        <west>-89.3833</west>
      </LatLonAltBox>
      <Lod>
        <minLodPixels>256</minLodPixels>
      </Lod>
    </Region>
    <GroundOverlay>
      <name>St Louis Bay</name>
      <Icon>
        <href>SuperOverlay/MSB_BATHY_G2.jpg</href>
      </Icon>
      <LatLonBox>
……
      </LatLonBox>
    </GroundOverlay>
    </Folder>
  </Document>
</kml>
```

Figure 2.14. Sample kml file (slb_u.c_region.kml) for generating the Regions in Figure 2.13. Note that all latitude and longitudes must be positive or negative. Mixed reference systems give unreliable results in Google Earth.

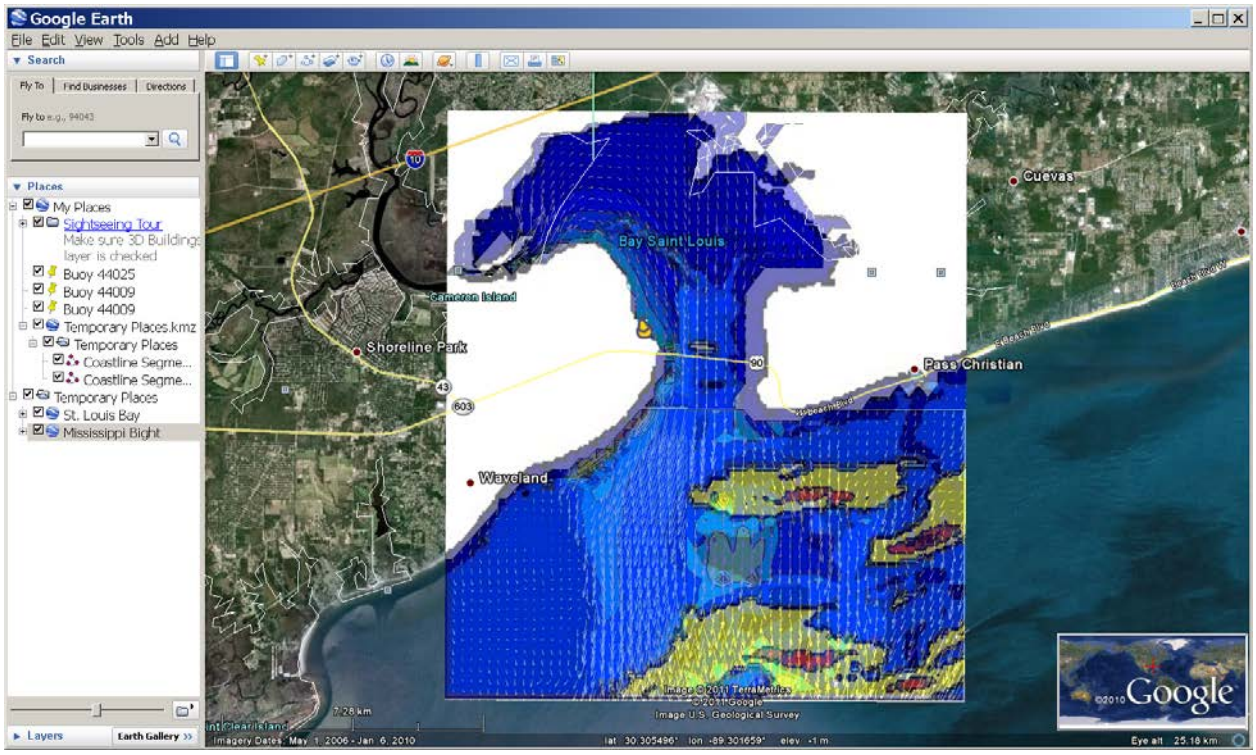


Figure 2.15. Screen shot showing overlapping images from St. Louis Bay. The model grid is shown as the lowest layer. Note the channel inside the bay as well as the deep areas within the sound. Above this are the contour plot of the tracer concentration and vectors for surface currents. The transparency is adjusted using the slider at the lower left of the window. It is adjusted to one-half for the Mississippi Bight layer (highlighted in the Places menu above the slider).

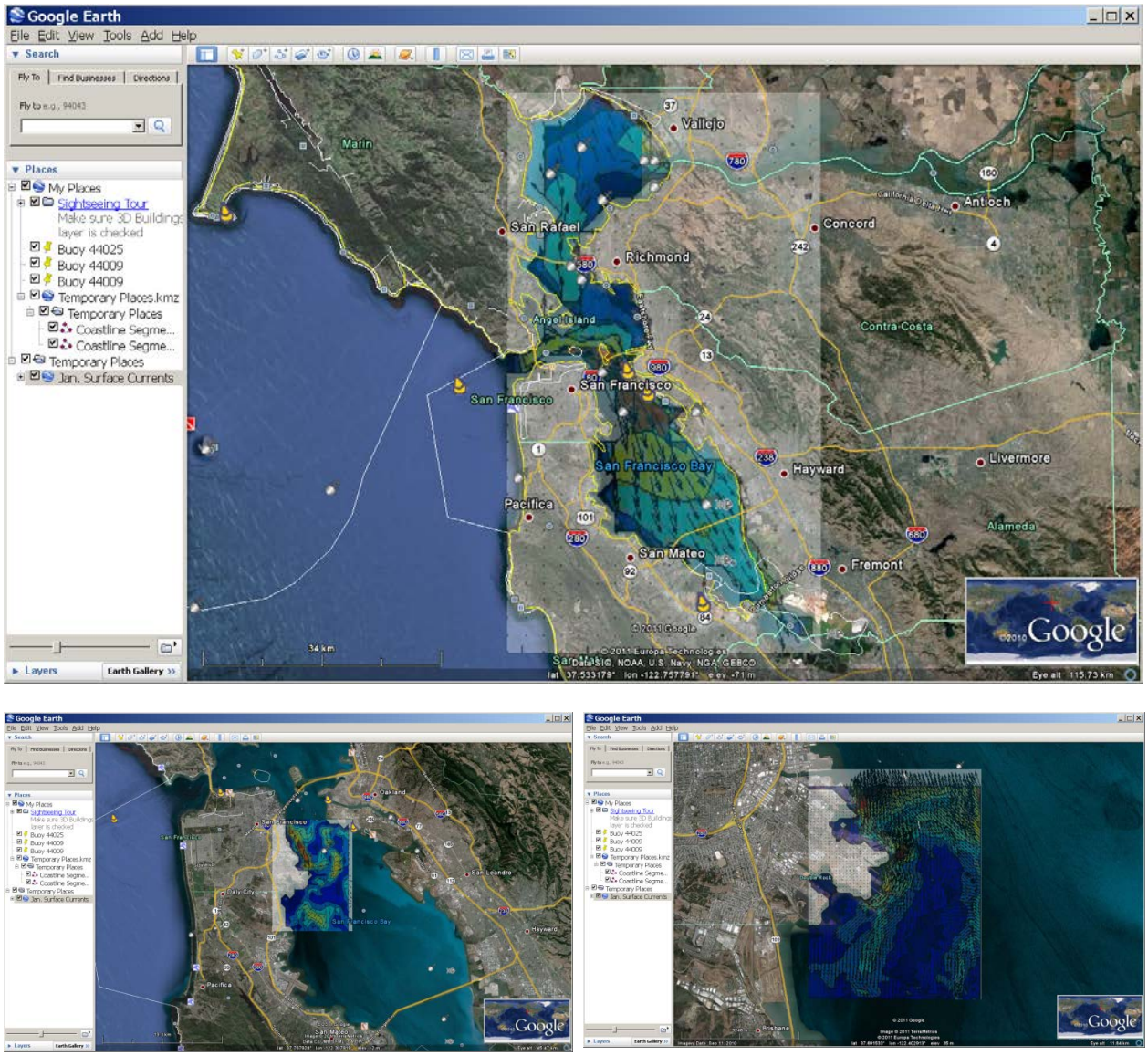


Figure 2.16. Screen shots of the San Francisco Bay nests. These Regions overlap as they refine the area around Hunters Point (lower left). However, even the highest resolution plot is insufficient to show all of the grid cells—it displays every third cell. This is why the extra plots from the St. Louis Bay simulation are necessary to see all of the data. These plots are much easier to make, however.

3. ASW Reach-Back Cell Oceanographic Analysis System (ARCOAS)

Background

Also known as the ASW Reach-back Cell Oceanography Analysis System (ARCOAS), this less generic name is rooted in the original requirement to support naval operations in anti-submarine warfare (ASW). At NAVO, the oceanographers and forecasters in the ASW Reach-back Cell (RBC) analyze ocean and atmospheric data to build a comprehensive picture of the environment in operational areas of interest. Various tools in MATLAB, on web servers, and other specialized software tools were used, adding to the complexity of the analysis process. The ever increasing volume of measurements in the ocean and atmosphere, and expanding numerical modeling prediction capability, called for a single, multi-purpose user interface with a concise set of software tools for the user to complete the analysis in an organised work flow. ARCOAS offers a certain environment and language in which the oceanographer is familiar and comfortable, while providing environmental information in a common geographic reference frame.

Allen et al. (2008) use the GIS (Geographic Information System) in a workflow approach in which model predictions and observational data are merged and visualized in a geospatial context for analysis. Here the conversion to GIS format of the data of interest is described as well as the implementation of standards for input and output of the map server used in ArcIMS (Arc Internet Map Server).

Software Approach

ARCOAS consists of ArcMap with customizations built into an add-in that includes a user interface and tools to manipulate data. ArcMap is a powerful software package using GIS technology developed by Environmental System Research Institute (ESRI), Inc. (http://www.esri.com/). A highly flexible programming language used in Microsoft applications, Visual C#, provides a rapid development path for invoking the GIS functionality of ArcMap coupled with other components such as the NetCDF (Network Common Data Format) library of functions and Zed Graph, a flexible graphics package. Thus, ARCOAS is designed and built with an object-oriented approach that incorporates listeners, dialog boxes, interactive displays, clickable buttons all put together to offer the user tools tailored to the work at hand.

Applications

The primary focus of the oceanographers in the ASW RBC is to assess the level and confidence in the data and model output in depicting the current and predicted structure of the ocean. One goal is to provide a level of certainty to other users of the data and model output as it affects their applications (e.g. algorithms for ocean acoustics) and their own interpretations on which critical mission decisions are made. The first step is to quickly locate the data with respect to the earth, a step at which a GIS is inherently quite proficient. Figure 3.1 shows a world map view rendering available domains from numerical models: (1) the global Navy Coastal Ocean Model (NCOM) run at NAVOCEANO; (2) the Coupled Ocean Analysis and Forecast Prediction System (COAMPS®) run at Fleet Numerical Meteorology and Oceanography Center (FNMOC) in Monterey, California; and (3) WAVEWATCH III™ run at FNMOC and NRL

Stennis. Once located, data can be visualized as shown in Figure 3.2 where the view is zoomed to a model domain to focus on the locations of the plotted profiles and the layer rendering the model output of surface water temperature. Other items rendered are locations of environmental measurements such as from expendable bathythermographs (XBTs) and Argo floats collecting ocean profiles of temperature and salinity; remotely sensed data tracks taken by radar altimeters on polar orbiting satellite platforms measuring sea surface height (SSH) variations; and multi-channel sea surface temperature (MCSST) data derived from brightness temperatures measured from satellites.

Although Figure 3.2 shows a myriad of observation points within this prospective region of interest, further analysis can be performed on specific observation data using a guided process of elimination and selection. Figure 3.3 shows a plot of profiles of measured and modelled, time-coincident parameters at a specific location for comparison. More interesting is the comparison of the overlaid sonic layer depths (SLD) computed based on the plotted profiles. SLD is highly sensitive to variations of the thermohaline structure which directly affects performance of acoustic systems supporting ASW.

Additional capabilities provided in the user-interface structure of ARCOAS include layer mathematics, animations, cross-sections, various drawing functions, a highly configurable set-up utility, exporting presentable graphics, and model performance statistics.

Functionality

The following key points make ARCOAS functionality unique. Interaction between map and menus allows for ease of selection of data and provides a means to narrow down the choices based on location and other pertinent information simultaneously. Accessing data sets without thinking about files takes away from the user the burden of interpreting what kind of data is desired though data discovery by file system search which can be sometimes quite cryptic and laborious to investigate. Spawning external programs or tool boxes is convenient because users can stay focused in one software environment, letting the computing environment to implement various software packages or algorithms. Interacting with data wherever they originated helps the user stay organised and on task, so that updates and changes are immediate, where they need to be, and without confusion.

Implemented using a GIS display and associated GUIs, users can display geophysical data derived from observations collected in a variety of ways, their locations on the map and information about that data in tables and graphs. Geophysical parameters made available in four dimensions can be displayed on the map in 2-dimensional layers that can be enhanced for visualisation. Typically, this form of data, i.e., “dataset”, typically comes from numerical model in the form of NetCDF files. Geophysical data for the purposes of supporting Naval operations, both observations and model predictions of oceanographic and meteorological parameters, include temperature, salinity, currents, wind, humidity, pressure, and, water elevations.

Interfacing with Data Services

Since ArcMap is the application on which ARCOAS runs, data services that are available to ArcMap are inherently available to ARCOAS, though the current version does not directly

access any data other than on the local file system, such as flat files, shapefiles, GRID files, netCDF and HDF files, and geodatabases. The broader data access made possible with ArcMap (and ArcGlobe and ArcScene for that matter, all described at http://www.esri.com/) also includes services such as ArcGIS, ArcIMS, Web Features Services (WFS), and Web Map Services. It is safe to say that whether the data are handled through a local geodatabase or a remote map server on the web, data accessibility could be transparent to the user. ArcCatalog is the accompanying data management application for users to manipulate data services or just browse the data.

The geodatabase is an object-relational database approach for storing spatial data serving as a "container" for holding data sets, which in legacy systems consist of shapefiles, raster files, and GRID files. Since the geodatabase can hold multiple feature classes in one place as well as raster layers and tables, it can provide a convenient method for keeping a project with topological information relating all the data contained therein. Data can also be stored or published at a service site in much the same way. Taken to the next level, geodatabases and geodata services can be managed by the ArcGIS server, which enables more concepts and capabilities.

Access to the data services as described above while using ArcMap (which could also mean while using ARCOAS) as the client can be done with the "Add Layer" button, which presents a menu of choices in the spirit of accessing any of the data in one place. The compliance with OGC standards makes this all possible in a general way. Maps and imagery as described above are examples. Other clients, such as ArcGIS Explorer (freely available GIS viewer) can work as a client for data services such as ArcGIS Server, ArcIMS, ArcWeb Services and WMSs, all describe at the ESRI web site. Furthermore, the ArcGIS Web Mapping APIs allow embedding into applications browsers as plug-ins to make them clients for accessing and manipulating geodata.

ARCOAS has specific functions that save the current project not only in an MXD file, but also puts together an accompanying geodatabase file as specified by the user. Since any given project may access multiple sources, from a remote service or the local file system, users cannot assume that all the data used in the current project will be available later. Therefore, users have to be aware of what data needs to be stored in the project geodatabase for later use.

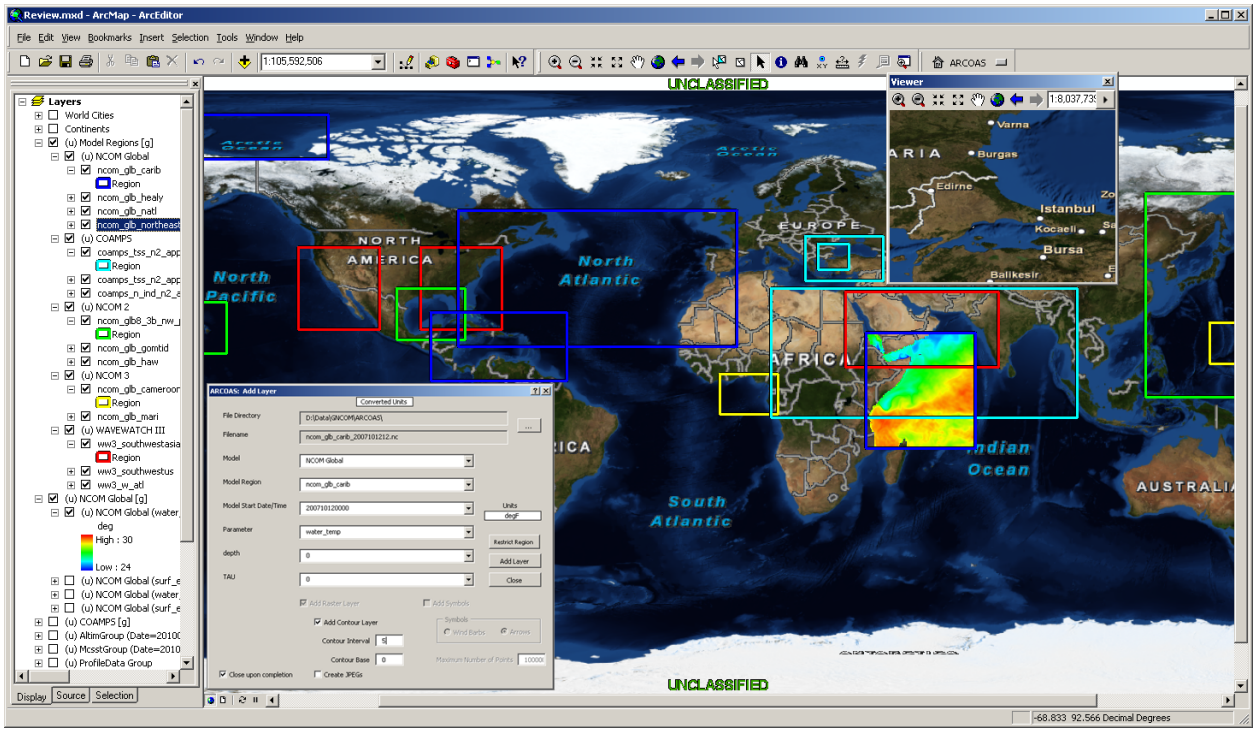


Figure 3.1. Global view of atmospheric and ocean prediction model domains available for analysis. The colour-coded boxes correspond to domains organized in groups in this case by type of model, including global NCOM subsections, COAMPS, and WAVEWATCH III. The table of contents on the left indicates what is available for display. The inset dialog box is the interface for bringing up a data layer from output of a model domain. In this case a region in the northeastern Indian Ocean was selected and a raster data layer of temperature at the surface from NCOM is displayed.

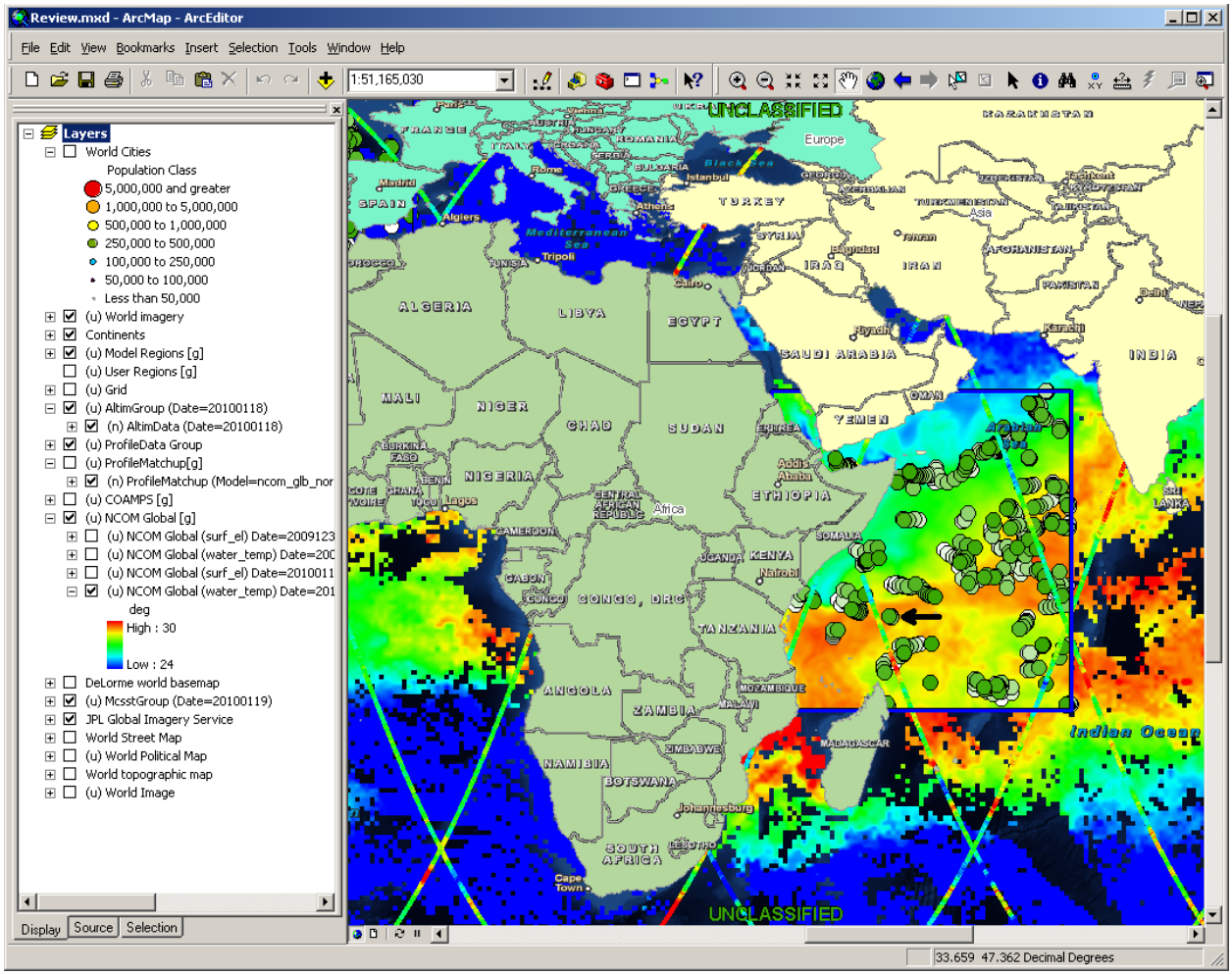


Figure 3.2. The raster layer of surface temperature from global NCOM in the northeastern Indian Ocean and the satellite-derived MCSST (see Figure 3.1), showing that they are fairly consistent with each other. Measured SSH points along tracks taken with a radar altimeter for one day and profile measurement points limited to the NCOM domain are plotted on the map as well. Greener points are newer. One selected point indicated by the arrow corresponds to plots in Figure 3.3.

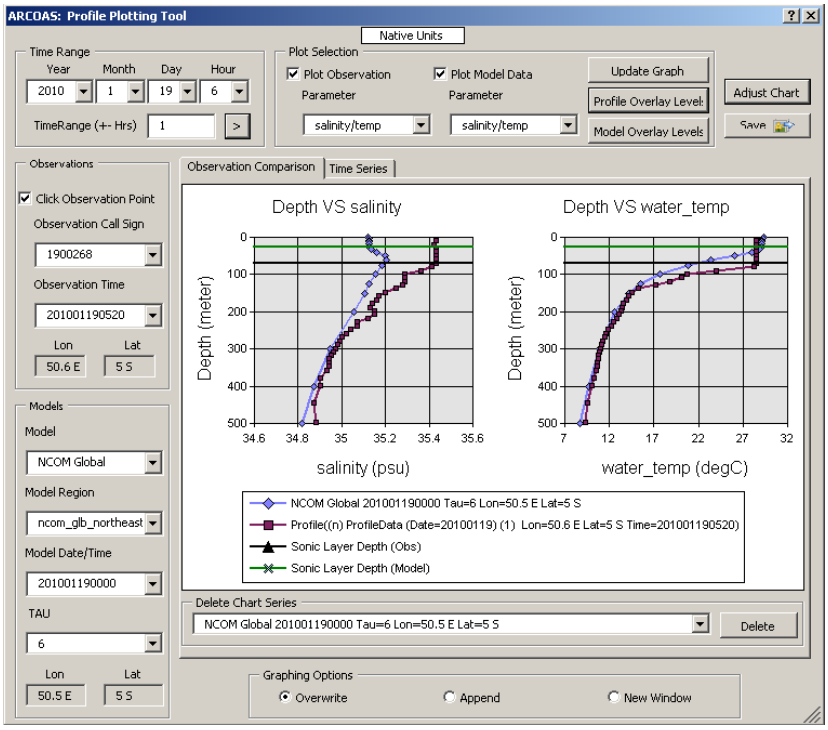


Figure 3.3. The point selected by the arrow in Figure 3.2 corresponds to this plot of the profiles of measured and modelled temperature and salinity. The SLD levels overlaid on the profile plots are calculated based on both measured and modelled parameters and show how sensitive SLD is to the ocean thermohaline structure.

4. GServer Web Service

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems (httpd.apache.org). It has been on the internet since April 1996. The project is jointly managed by a group of volunteers located around the world. Actions and errors generated by *Apache* web server for *gserver* are written to the following files:

```
/var/log/httpd/gserver-access_log
/var/log/httpd/gserver-error_log
```

These are useful for debugging the Web server and cgi scripts. The prototype Web Server being used in this work is

```
http://gserver:8000/Home_Page_for_Google_Earth_Project.Beta.htm
```

A Web Server can be made to respond to special requests from a client, even just using wget, in order to obtain certain types of data offered by the server. For example, a server at FNMOC was developed to make available spectral data at every grid point of the globe as it was output by a global ocean wave model. A well-contrived http request in a broker language that is interpreted at the server would cause software at the server to produce the data in some format ready for the delivery to the web client. This server side software could be a data collector of some sort that simply queries the data base (or just a file system of files) and assembles files in an expected format. It could use the request in which the user-specified points form the basis for a model run that will include these locations in a model domain and provide the output in the expected format with some additional post-processing.

Generating Forms with CGI scripts

This demonstration is focused on the *Regional Simulation Database* button from the main menu of Gserver. All of the databases that show up under Gulf of Mexico are from Gerris. The available databases are listed in file, **database.gv**, which is currently located in:

```
http://gserver/Databases (/home/keen/GoogleVision/Databases)
```

The entries are: (1) region; (2) database name; (3) longitude; (4) latitude; (5) file system path; (6) date (YYYYMMDDHH): (7) time (MM:SS); (8) domain size (km); (9) number of variables; and (10) a list of space-separated variable names. These are used to search the databases by the cgi scripts. This discussion focuses on example scripts listed below (Located in /home/keen/GoogleVision/SCRIPTS):

```
Home_Page_for_Google_Earth_Project.Beta.htm
form_get_region.py
form_get_databases.py
form_read_msb_gfs.py
```

```
form_plot_variable_msb_gfs.py
read_msb_gfs.py
read_databases.py
interp_idw.py
```

The process begins on the GoogleVision home page (Figure 4.1). This is an html document, but selecting the *Regional Simulation Databases* link instantiates the *form_get_region.py* cgi script using the following line:

```
...
<tr><td><a         href="http://gserver/cgi-bin/form_get_region.py">Regional
Simulation Databases</a></td></tr>
...
```

The name of this script follows a useful convention of beginning with the keyword, *form*, to indicate that it is used to create an html form for sending to a web browser.

This is a simple form with two components: (1) a drop-down list of available regions that contain accessible databases read from the file, **databases.gv**, and (2) an action bar labelled "List Available Databases". This script consists of print statements that are interpreted by a web browser using the *Content-type:text/html* tag. Sections of this script are shown below to demonstrate the general form of the cgi script that creates a form for interacting with a browser:

```
#!/usr/bin/python
from numpy import *
import os, sys
import cgi, cgitb
from read_databases import *
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Model Simulation Databases</title>"
...
print "<body>"
...
numrg = Read_Regions (region)
numdb = Read_Databases (dbname, dbpath, dbdtg, dbnvar)
for dbi in range (numdb):
      index = dbi
      Read_Variables (index, var, dbnvar[dbi] )
print "<form action=\"http://gserver/cgi-bin/form_get_databases.py\">"
...
print "<select name=\"region\">"
while cnt < numrg:
      print "<option value=\"%s\">%s</option>" % (region[cnt], region[cnt])
      cnt = cnt+1
print "</select>"
print "<input type=\"submit\" value=\"List Available Databases\">"
...
print "</body>"
print "</html>"
```

Note that a backslash is the escape character that indicates the the double quotes inside the string are characters and not string delimiters. This script sends a sequence of *print* statements to the browser, which are interpreted as an html document that produces the page to request all available databases for a selected region (Figure 4.2). These include “Gulf of Mexico” and “Arabian Gulf”. Only the Gulf of Mexico is currently active. When this region is selected and the *List Available Databases* button is selected, the *form action* tag from the file above is invoked and the cgi script, *form_get_databases.py', is instantiated.*

This form (Figure 4.3) reads the **databases.gv** file and lists all of the databases for the selected region in a drop-down box. The labels in the box include the database name, location (longitude, latitude), size of the simulation domain, and the date assigned to the results. These details are intended to help select a database. Clicking on the *Open* button invokes the *action* associated with the form, which in this case executes the cgi script, *form_read_msb_gfs.py*.

Information from a form is entered in one of the boxes as shown. This request is passed to the invoked cgi script through the QUERY_STRING environmental variable. This string includes the entire request sent to the server from the client browser (e.g., “database=Mississippi_Bight_GFS-OM004&variable=current&lat=30.4&lon=271.8”). The parts of the string are read from the html lines within *select* and *input* tagged sections from the script, *form_read_msb_gfs.py*:

```
...
print "<select name=\"database\">"
print "<option value=\"%s\">%s</option>" % ( database, database )
print "</select>"
...
print "<select name=\"variable\">"
while v < len(var):
      print "<option value=\"%s\">%s</option>" % ( var[v], var[v] )
      v = v+1
print "</select>"
...
print "<label>Latitude:  <input name=\\"lat\\" size=\\"12\\"></label>"
...
print "<label>Longitude: <input name=\\"lon\\" size=\\"12\\"></label>"
 ...
```

This code accompanies the page to select a variable and location to extract from a database (Figure 4.4). The *while* loop prints the available variables read from the **databases.gv** file into the list box. This form has four user-selected entries before it is submitted to invoke the script, *form_plot_variable_msb_gfs.py*, which is not actually a form at this time. It will become one when a file download feature is added.

The three scripts listed at the top of this section that do not begin with *form* are modules that include helper functions for accessing the databases and interpolation. Individual databases will probably require a form and possibly a module (e.g., *form_read_msb_gfs.py*, *form_plot_variable_msb_gfs.py*, and *read_msb_gfs.py*) to be supplied by their originator. There

must be a cgi script for each database because of differences in the data formats. This may be unnecessary for some model output but it does allow for standardization of the database access procedure. This places the responsibility of accessing the data accurately on the *producer* of the database rather than every person (or program) that accesses it.

The *form_plot_variable_msb_gfs.py* cgi script does not generate a form. Instead, it returns the requested information to the browser. The currently available data consist of interpolated time series of values for the selected lat/lon from all of the files in the database. The interpolation procedure is in a module, *interp_idw.py*. The time series of requested variable is written to a file in the *GoogleVision/download_tmp* directory. This file has the form:

```
$DATABASE_NAME-$VARIABLE_NAME_lat$LATITUDE_lon=$LONGITUDE.txt
```

The name given to this ASCII file is supplied along with the database reader script. This capability will be implemented using a file download script in the future.

```
Mississippi_Bight_GFS-OM003_surf_el_lat29.4_lon271.8.txt
```

Maps of variables are commonly used for both model development and operational simulations. These will be implemented using scripts to make color fill images of selected variables for regions/vertical slices/times. In order to test this potential plotting function, we have completed preliminary tests for handling this within the database cgi scripts themselves. The more standard approach is to use libraries and complete plotting packages to process the files. It might be easier, however, to include more primitive functions directly in the cgi scripts if *dynamic* queries and responses are to be used.

Image Generation in a CGI script

A test was conducted to see how much effort is expected to get a demonstration effort working. There are two approaches to passing an image to the client: (1) save the image as a file and pass it; or (2) stream the image directly without the added steps of creating and storing a file.

First we examine approach (1). This test uses an image of a color test pattern directly written by the program, **make_ppm.c**, which writes a simple set of color triples to a file in the **ppm** format. This format is not accepted by the browsers so it must be converted to an acceptable file type using the *system* ("convert") function. The original red-green-blue test pattern from this file (not shown) is poorly reproduced by the jpeg (Figure 4.5A) but the gif conversion (Figure 4.5B) is very accurate. There was an error in attempting to upload the **gif** file to this Wiki because it was apparently corrupted and showed no image. I opened the screen dumps in Powerpoint and saved them both (the **jpg** and **gif** files) as **jpg** files. This resulted in good transformation of the originally corrupted **ppm** file from the *convert* program. This needs to be examined further. An error (Figure 4.6) occurs when the image file is in the cgi-bin directory but it may be due to other reasons as well.

A test was completed for passing an image file to the browser using an *image* content type. This works as long as the image is not in the *cgi-bin* directory. This has been successfully implemented. The result (Figure 4.7) can be compared to the direct result (Figure 4.1). This capability will be added to the database cgiscripts in addition to the time series extraction. The

conclusion for method (1) is that we can *dynamically* produce images and pass them to a client. This method has been implemented in the script, **form_plot_variable_msb_gfs.py** using the *Matplotlib* utilities. This is very similar to Matlab but with fewer lines of code. Combined plots of two variables (e.g., *u* and *v* currents) are plotted (Figure 4.8), saved, and displayed in the browser with the following Python code:

```
 import matplotlib
 matplotlib.use('Agg')
 import pylab
 ...
 pylab.plot(hr, tsu)
 pylab.plot(hr, tsv)
 pylab.xlabel ('Hours after ' + dtg[0])
 pylab.ylabel ('M/S')
 pylab.title ('East (blue) and North (red) Currents at ' + latstr + ' N, ' +
lonstr + ' E')
 filename = top + database + '_' + variable + '_lat' + latstr + '_lon' +
lonstr + '.png'
 pylab.savefig(filename)
 if test == 1:
       pylab.show()
 filename = database + '_' + variable + '_lat' + latstr + '_lon' + lonstr +
'.png'

 print "    " % ( '/download_tmp/', filename )
 ...
```

The second method is more desirable because of fewer steps to program and keep track of. It also does not introduce potential collateral problems associated with file access on the server. This requires some modification of the plotting procedures to pipe the output as raw data to the browser.

Ocean Data Mining

Serve the data from a static data base or request data creation on the spot—that is the question. The answer depends on the over-riding concern, saving time or space? A static data base of spectra produced ahead of time by a model such as WAVEWATCH III can be output in a file containing a snapshot of spectra for every grid point with grid spacing of 0.5 degrees from as far south at 70 degrees S and and far north as 65 N. This snapshot (restart) file can be used to store the spectral state of the model for the entire domain but it is about 535 MBytes. For a day every hour the volume of data adds up to 12.6 GBytes. One year of model output would add up to 4.5 TBytes.

The Global NCOM data base is a good example of the difficulty of dynamically accessing large files using a web server (Figure 4.10). A plot of sea surface height (see Figure 4.11) was completed within the allocated time-out but the currents for one year cannot be accessed in a timely manner because of the need to unzip them. Python can read a zipped file using the *Gzip* module but it has to write the data to a temporary file to access them using the NetCDF4 functions. There are also twice as many data values for vector data.

To run this model dynamically from nothing but an archive of winds fields from a global atmospheric model such as NOGAPS requires that the wind input be assembled for an ample

period of time before the time of interest for a sufficient spin-up. A period of a week for a given domain would require that the global model spin up for a week ahead of time. Total run time could take in the order of about 10 minutes, post-processing a few more minutes.

A happy medium might be the storage of periodic restart files, say one per week, so that you would never need more than a week restart, and all the data from the start is not part of a spin-up. This is reasonable to satisfy most hindcast requests.

This question has been investigated with respect to extracting basin-scale results (N. Pacific) for decadal periods. The resulting subsampled fields of surface currents from the 1/8° NCOM global fields have been used to study the distribution of flotsam in the N. Pacific (aka Great Pacific Trash Vortex). The fields were applied to a particle tracking model.

DATABASES

The databases are added to this list as they are made available; i.e., as reader software is written and incorporated into the GServer web page. The permanent databases that are included are described in this section.

Global NCOM 1/8°

Table 4.1 summarizes the contents of the database. The files are contained within a series of subdirectories under the top folder, /u/NCOM:

```
eas16_tides- model geometry netcdf files; no results
glb8_2a-     model geometry netcdf files; no results
glb8_2b-     years 1998-2001
             daily files
             compressed (gz) netcdf files for ssh, sss, sst, ssu, and ssv
glb8_2c-     years 1998-2002
             daily forecast files (τ = 0-168 hour)
             netcdf files for ssh, sske, sst, sss, ssv, ubh, and vbh
glb8_2f-     years 1997-2008 in subdirectories (see Table 1)
glb8_3a-     30 Aug 2007 (nowcast and 72 hour forecast)
glb8_3ac-    years 2006-2008 in subdirectories
             daily nowcast files (τ = 0 hour)
             netcdf files for ssh, sss, sst, ssu, and ssv
glb8_3b-     years 2008-2012
             3-hour netcdf files
             netcdf files for ssh, sss, sst, ssu, ssv, s3d, t3d, u3d,v3d
glb8_3b_hc-  years 2007-2008
             3-hour netcdf files
             netcdf files for ssh, sss, sst, ssu, ssv, s3d, t3d, u3d,v3d
```

This list is a brief summary only. There are a number of other databases that are not on this file system. These are discussed in the following document:

`/home/fitzgrld/ncom1/`**`README.NCOM_CATALOG`**

Some of these have a reading script/program available but they are not part of a database at this time. This section will focus on only those (see Table 4.1) that are relevant to the North Pacific trash vortex study summarized as:

- NCOM_glb8 (0.0 E, 0.0 N): Size = 40075: Date = 1997070100 (1 July 1997 to 29 Aug 2008 12:00) /u/NCOM/glb8_2f/nc

Global Ocean Observations

The ocean observations consist of satellite, ship, glider, and buoy data from many locations. These data are available locally at the following location:

```
/u/prob/ncoda/data/navoqc
```

The variables consist of:

```
altim:    altimetry
goes:     geostationary operational environmental satellite
lac:      NOAA local area coverage
mcsst:    multichannel sea-surface temperature
profile:  expendable bathythermographs
          First file is 1 Nov 2007
ship:     ship-board observations
ssmi:     special sensor microwave imager
```

These are stored in subdirectories under *navoqc_public/*. These data files are read by a Fortran program (**ocn_obs.f**):

```
/u/LBSFI/spence/ocn_obs_reader/ocn_obs
```

Here is an example use:

```
[user]$/u/LBSFI/spence/ocn_obs_reader/ocn_obs profile 2007110900


   ****** Reading PROFILE Data ******
 file date time group: 2007110900
  data directory path: /u/prob/ncoda/data/navoqc/navoqc_public
      number profiles:       1367
    max number levels:        561
  file version number:          3
-------------------------------------------------------------------------------
profile call sign           : "22522  "       1
profile latitude            :         8.95
profile longitude           :       108.46
profile observed DTG        : "200711090148"
profile received DTG        : "200711090404"
DBDBV bottom depth          :         93.0
profile data type codes     :      5    33
temp data type              : "  Drifting BUOY Temp"
salt data type              : "      MODAS Salinity"
observed temperature levels :           10
observed salinity levels    :            0
temperature gross error     :       1.0000
salinity gross error        :       1.0000
sea surface height anomaly  :    -999.0000
sea surface temperature     :        26.52
security classification     :          "U"
   depth      temp   clm_std   tmp_err   tmp_prb   clm_tmp   mds_tmp
```

```
     13.0   -999.00      1.19      0.06  1003.000     26.82     26.48
     18.0   -999.00      1.16      0.06  1003.000     26.76     26.45
     23.0     -0.08      1.14      0.06     1.000     26.68     26.34
     28.0      0.04      1.16      0.06     1.000     26.56     26.19
     33.0   -999.00      1.18      0.06  1003.000     26.44     26.05
     38.0      0.04      1.22      0.06     1.000     26.29     25.92
     43.0      0.20      1.29      0.09     1.000     26.12     25.79
     48.0      0.08      1.36      0.10     1.000     25.95     25.67
     53.0      0.16      1.42      0.10     1.000     25.77     25.52
     58.0      0.12      1.51      0.11     1.000     25.52     25.35
 ------------------------------------------------------------------------------
…continued columns:

    glb_tmp   rgn_tmp   glb_std   rgn_std   tmp_xvl   tmp_xsd
      26.58   -999.00      0.37   -999.00     26.87      1.18
      26.59   -999.00      0.35   -999.00     26.81      1.16
      26.54   -999.00      0.34   -999.00     26.74      1.14
      26.48   -999.00      0.34   -999.00     26.64      1.16
      26.39   -999.00      0.33   -999.00     26.52      1.18
      26.23   -999.00      0.32   -999.00     26.40      1.22
      26.16   -999.00      0.38   -999.00     26.25      1.29
      26.12   -999.00      0.52   -999.00     26.11      1.35
      25.95   -999.00      0.72   -999.00     25.93      1.41
      25.67   -999.00      0.97   -999.00     25.59      1.50
     ----------------------------------
```

This program can be used to accumulate data to validate the ocean model database from NCOM. This may require modification of the source code, however, **ocn_obs.f**. There is an alternative file, **ocn_obs_new.f** but this had a memory error when compiled with gfortran. More later...

The following Matlab scripts also access these databases:

```
/home/rowley/matlab/ocnqc

load_altim_qc.m
load_mcsst_qc_simple.m
load_ncoda_qc_profile_full.m
load_ssmi_qc.m
```

WEB SERVER SOFTWARE

The underlying method is to implement reading software through the GServer web page (gserver:8000). This is being done as discussed for the web server documentation. There is some redundancy with that document and the reader should check both. The opening page to the GServer web server is:

```
http://gserver:8000/Home_Page_for_Google_Earth_Project.Beta.htm
```

This is the page that is being actively developed at this time (Figure 4.1). Selecting the *Regional Simulation Databases* button opens a Python cgi-script (*form_get_region.py*), which sends a form to the browser. This form has a drop-down list from which the *Global* database is selected. When the *List Available Databases* button is selected, the cgi-script, *form_get_databases.py* is invoked and creates a new form that lists the global databases. An example is the first database listed above. The name is based on the request to the form from

the previous form. The coordinates are general; in this example, they indicate it is centered at the intersection of the equator and prime meridian. The size is the breadth of the domain, which in this example is the Earth circumference. The date is the starting date for the database. When a database is selected, the *form_read_Global.py* cgi-script is called (Figure 4.10).

Figure 4.10 shows the form for selecting a single point for data extraction. When a variable and location/time selections are made, the cgi-script, *form_plot_variable_NCOM_glb8.py* is called as the action. This script actually accesses the database and creates the requested output. An example can demonstrate the output from this form. We can plot *SSH* for the period 1998-2002 at longitude 180.892 E, latitude 29.4 N (Figure 4.11). The straight line in the figure represents bad files from the database, in this case SSH exceeded 2000 m at this location.

DATABASE ACCESS

Each database is unique. No standards have been applied to the output. Most of those available do use NetCDF files but without standard variable names or output formats. This necessitates very specific cgi-scripts to read them. Ideally, these scripts will be supplied by the creator of the database. This is not the case. The global NCOM database is described as an example of the types of issues to be aware of in constructing a web-accessible database and its required reader software. The variability of this database can be seen in Table 4.1.

There is substantial variation in the data availability for multiyear scales. This makes the reader more difficult to write. These variations must be incorporated using *Exception* catching. As an example of this method, consider the following Python code for reading a packing attribute name from a NetCDF file, which may not be present in the file/database. Some of the global data are packed and some are not.

```
nc = Dataset (ncfile, 'r')
while True:
       try:
              uDataMin = nc.variables['U_Velocity'].pack_min[k]
              uDataMax = nc.variables['U_Velocity'].pack_max[k]
              ubits = nc.variables['U_Velocity'].pack_nbits
              break
       except AttributeError:
              print "Attribute not found: File not packed."
              break
       ncU = (nc.variables['U_Velocity'][k, :, :] )
       nc.close()
```

There are several steps to complete to successfully read a database entry. There is one entry for each time within the requested interval. The current configuration for the global databases uses only fields for 00:00 on each day. These are not averaged. These are listed in **dtg.YYYY** files in /home/keen/GoogleVision/Databases/NCOM_glb8. The model output levels are also listed in files, **zm.txt** and **model_zm.nc**.

1. Find the file containing the variable, which may be in one of several directories
2. Uncompress any files as indicated in Table 1
3. Find the requested variable in each file
4. Fetch the variable from the file
5. Interpolate to the requested location

File exceptions can occur at steps (1)-(4). Exception handling can make the variable extraction easier to code. For example, Figure 4.11 shows 6 years worth of SSH but no data were read for approximately days 1200-1700. The cgi-script prints an error file with all bad data points listed. This file is important for identifying valid fields and understanding their causes. A typical ERROR file is:

`download_tmp/NCOM_glb8__ERROR_ssh_Surface_lat29.4_lon217.8_1997-2002.txt`

This file shows that values of *SSH* exceed 2.7×10^3 on 17 May 1998, 31 Aug 1999, and 357 times between 20 Jun 2000 and 30 Sep 2002. The long string of bad values in Figure 4 are associated with continuous excessive SSH between Feb and Dec 2001. This is not a problem of the web server, however—*let the user beware*.

The extraction steps above are written into the database retrieval software. The directories are constructed using the top directory and any date-dependent sub-folders (e.g., "1998"). Only 3D files are compressed. These can be accessed directly by NetCDF 4 in Python but the individual variables cannot be extracted without writing out the contents to a temporary file. These files are written to /tmp on the server host (e.g. *typhoon*).

The initial purpose of this work is to extract surface currents. This can be done by either retrieving them from the **u3d** and **v3d** files or the **ssu** and **ssv** files (see Table 4.1). The latter is preferable because it skips step (2), which is time consuming, but not all years contain daily fields of surface currents; these variables are only available from 2001 through 2008. Nevertheless, this approach will be used for the time being until the usefulness of the additional 3.5 years can be determined. These files are all uncompressed and the frequency is a standard 3 hours. However, we will continue to use the daily values from the database listings.

DATABASE PRODUCTS

The database consists of a number of different variables like those listed in Table 1. These are useful of themselves for evaluating the ocean state predicted by the model. This instantiation of the ocean can be compared to others (e.g. satellite images). There are several ways in which ocean instantiations can be constructed from the data base: (1) time series of model variables at specified locations; (2) maps of variables at different depths; (3) cross-sections of lat/lon and depth, or random orientations; (4) phase plots of lat/lon/depth versus time. These products can be used to generate derived fields that can be used for problems like the Pacific Trash Vortex. The previous section has demonstrated the time series plots of SSH at interpolated points (Figure 4.11). The same can be done for the surface currents.

The cgi-script, **form_plot_variable_NCOM_glb8.py** extracts a subdomain using the menu (Figure 4.12A), which contains an input box for the requested output domain. The input data are output as **csv** and saved to the download directory below the GoogleVision home (e.g /export/home/typhoon/keen/GoogleVision/download_tmp/BOXFILES). The format of these files includes the variable name (e.g., ssu), depth (e.g., Surface), lat and lon, box size (e.g., 0.5), and input file date (e.g., 2001031700):

`NCOM_glb8_ssu_Surface_lat29.4_lon217.8_box=0.5_2001031700.csv`

The requested lat/lon is the center of the region and the box is centered on this location (Figure 4.12B). The file contains longitude, latitude, and eastward and northward surface currents (e.g, for *ssu*). An example of a map is the 30-day average surface currents for the N. Pacific (Figure

4.13). These plots are sent directly to the browser as described above and saved as files. The original values are written to files as shown above.

The vector plot output in Figure 4.13 suggests that there is a maximum resolution that is useful for our purpose. We can use this example to estimate the output cell size from the number of cells displayed in the vector plot: 10° ÷ 0.125° = 80 cells. It is possible to reduce requested output to be <100 cells along any axis. This may be unnecessary, however, because the user can potentially zoom in to clear up the images as needed.

The reduction of model output is being implemented using the Python arrays, *uavg* and *vavg*, which include subsampled output on a reduced region. The cgi-script *form_plot_variable_NCOM_glb8* is designed to work from the command line by manually editing to make the variable, *test* = 1. This is much faster, especially when there are file-system latency issues. The result from this procedure is much clearer to read (Figure 4.14).

The effect of the subsampling is seen in Figure 4.15. This indicates that care must be taken in specifying the output resolution or the result may not be suitable for the intended purpose. It would also be useful to have an accurate maximum current printed in the title for each plot because they do not use a standard vector length scale. The values in the NetCDF files also do no reflect those used in the subsampled plots. The final preliminary output for this cgi-script is shown in Figure 4.16. Note that the current vectors are given in cm/s. These scripts, **form_read_Global.py** and **form_plot_NCOM_glb8.py**, can be modified for other variables. They also need to be broken down into reusable code.

Tables and Figures

Table 4.1. File/output characteristics of the NCOM Global 1/8 ° data base for 1997-2008.

Year(s)	Variables	2D/3D Output Frequency (hr)	Compressed (2D/3D)	Notes
1997	ssh*, s3d, t3d, u3d, v3d	6/24	n/y	*24 hr frequency through 30 Jun
1998	ssh*, sst, ssu[%], ssv[%], s3d, t3d, u3d, v3d	6/24	n/y	* 3 hr frequency ; [%] 1 month frequency
1999	ssh, sst, s3d, t3d, u3d, v3d	6*/24	n/y	NA
2000	ssh, sss, sst, s3d, t3d, u3d*, v3d*	3/6	n/y	* 12 hr frequency
2001-2002	ssh, sss, sst, ssu, ssv, s3d, t3d, u3d, v3d	3/12	n/y	NA
2003	ssh, sss, sst, ssu, ssv, s3d, t3d, u3d, v3d	3/6*	n/y[#]	* 12 hr frequency before 1 Jul and after 1 Oct 12:00; [#] uncompressed for 1 Jul - 1 Oct 12:00
2004	ssh, sss, sst, ssu, ssv, s3d, t3d, u3d, v3d	3/3[@]	n/n[#]	@ 6 hr frequency before 1 May and between 1 Jun and 1 Sep; 12 hr frequency from 1-31 May; [#] compressed for 1 Feb - 30 Apr 18:00
2005	ssh, sss, sst, ssu, ssv, s3d, t3d, u3d, v3d	3/3	n/n	NA
2006-2008	ssh, sss, sst, ssu, ssv, s3d, t3d, u3d, v3d	3/6	n/n	NA

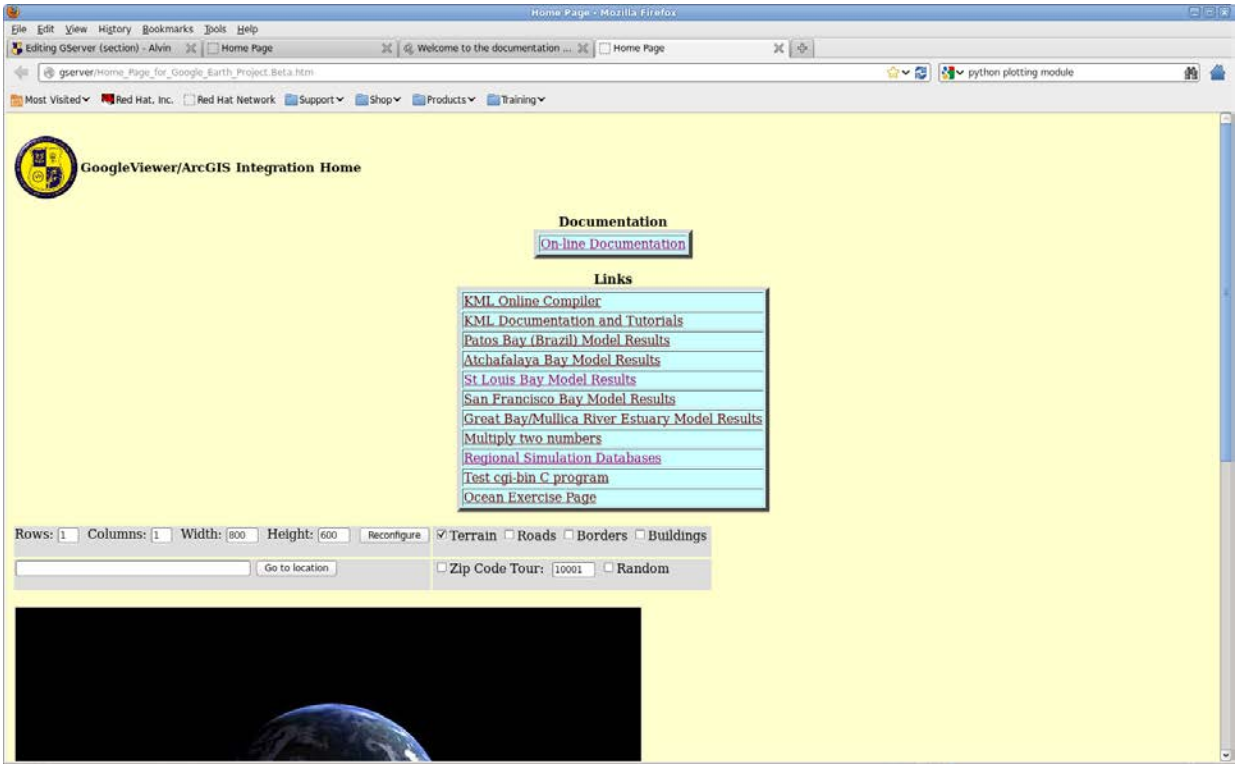


Figure 4.1. Screen dump of the GoogleVision home page. The button for finding a model database is labelled "Regional Simulation Databases".

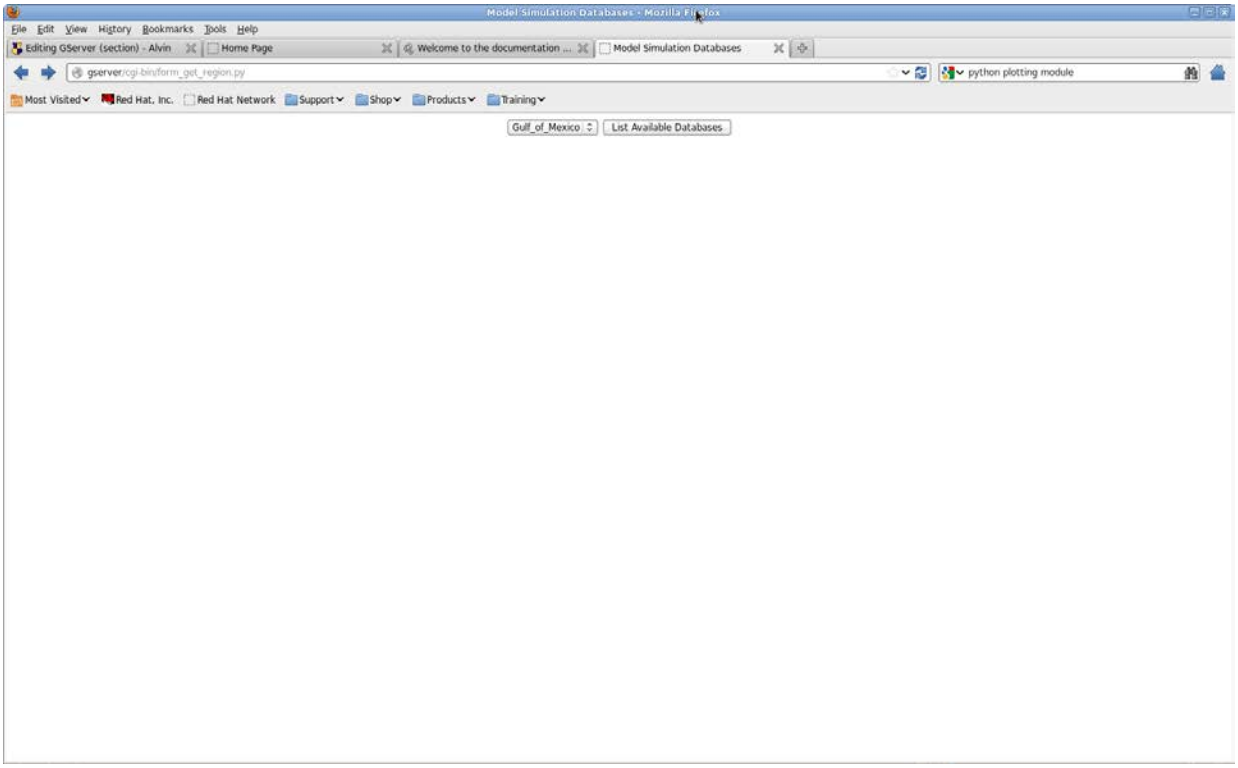


Figure 4.2. Screen image of the html form for selecting a model region.

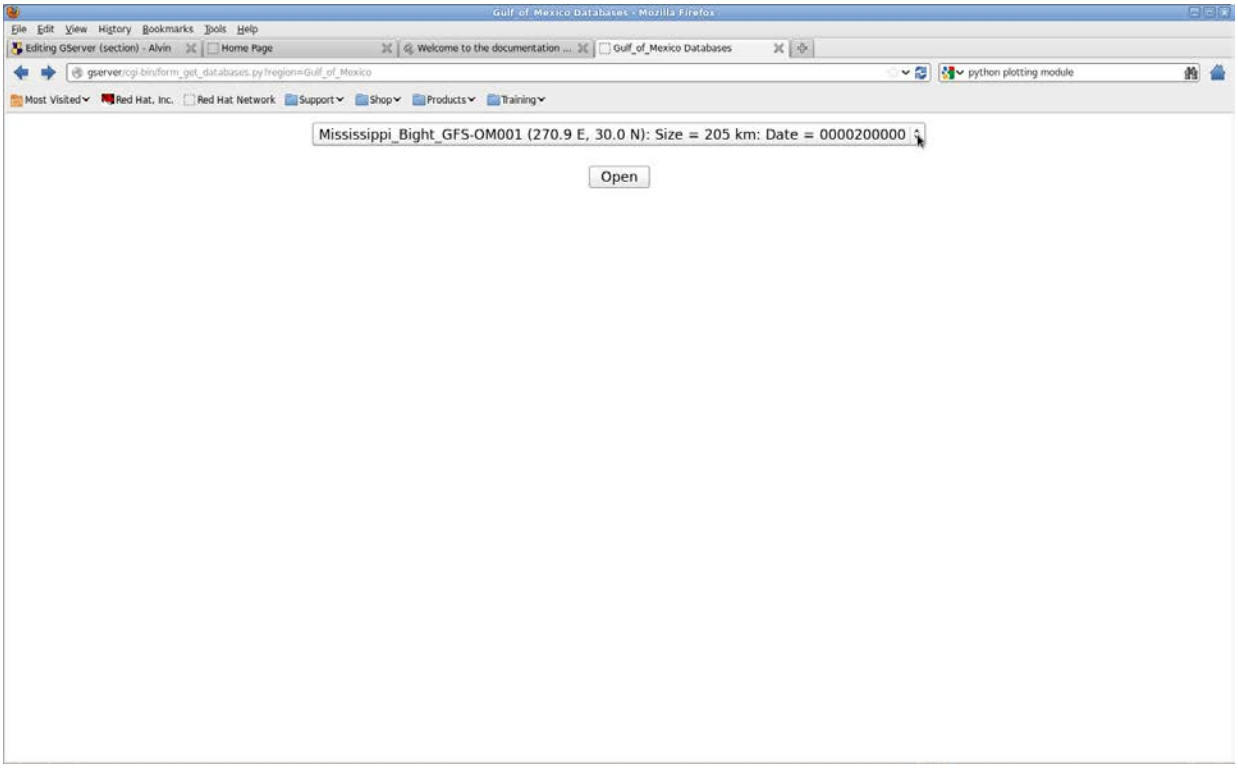


Figure 4.3. Page for selecting a database from the Gulf of Mexico region.

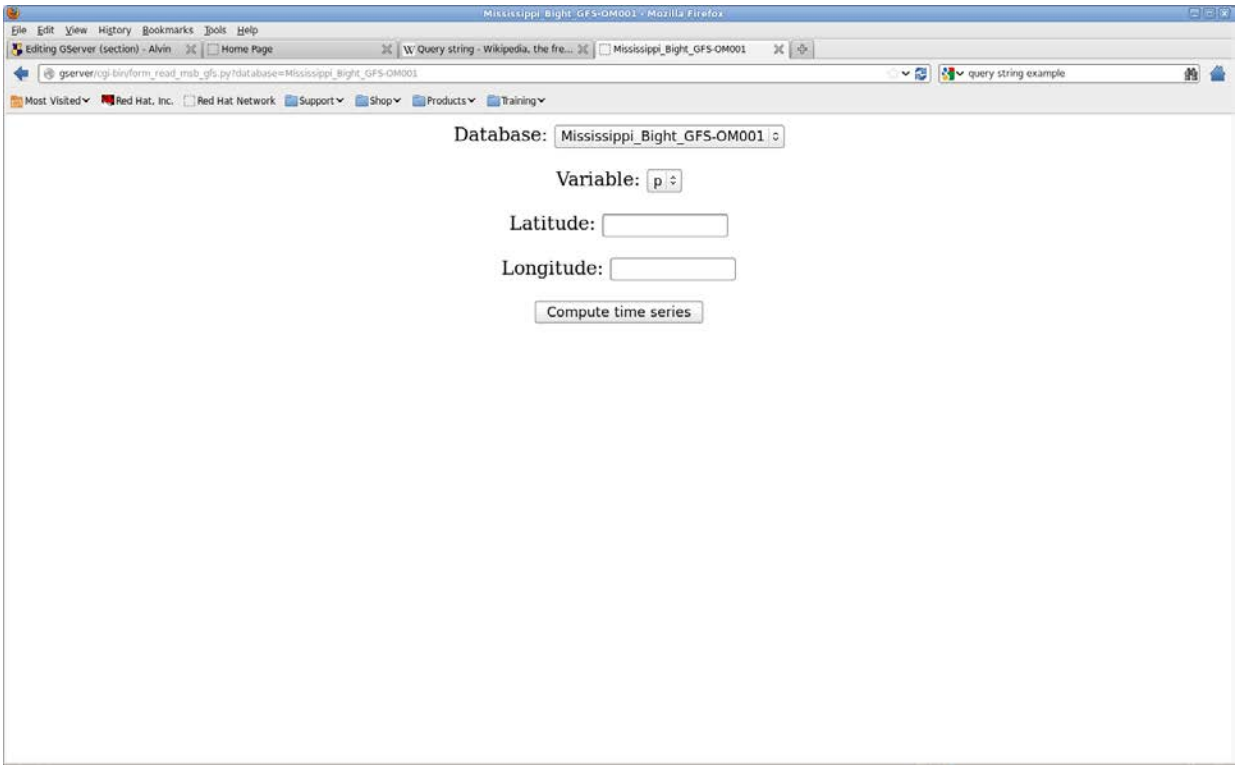


Figure 4.4. Form created by script *form_read_msb_gfs.py* to interpolate a set of fields to a time series.

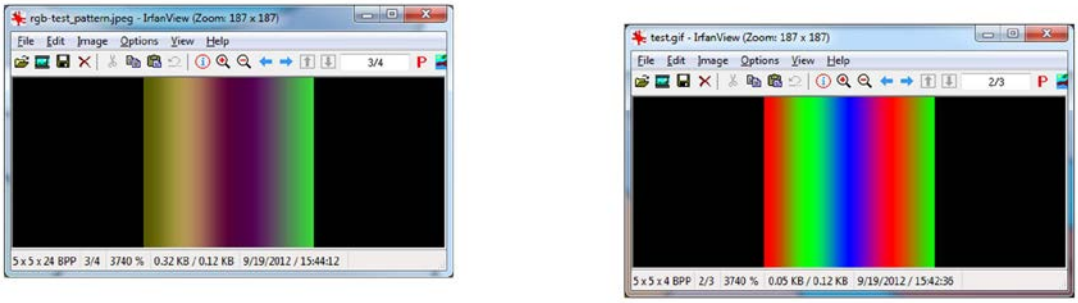


Figure 4.5. A. Test pattern converted to a jpeg file. **B.** Test pattern converted to a gif file.

Figure 4.6. Error from the browser when it attempted to show the images from Figure 1.

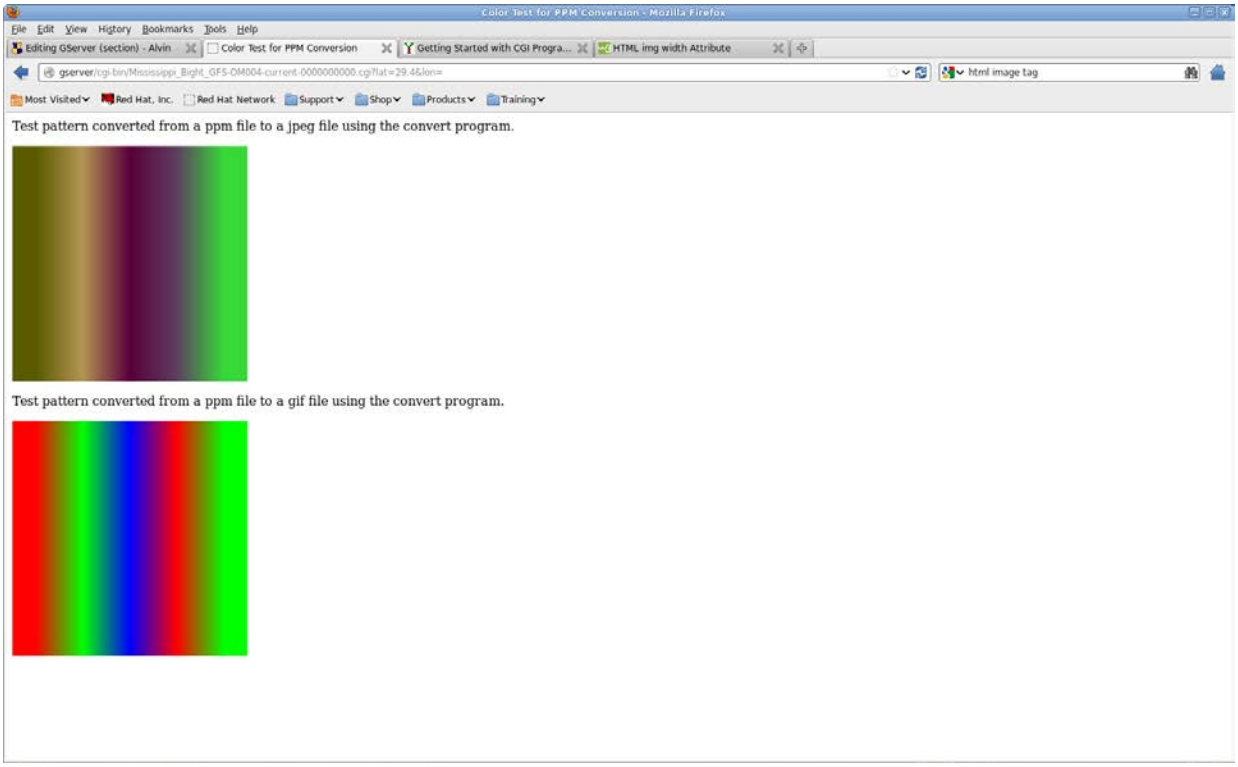


Figure 4.7. Screen dump from gserver web site for the images from Figure 1.

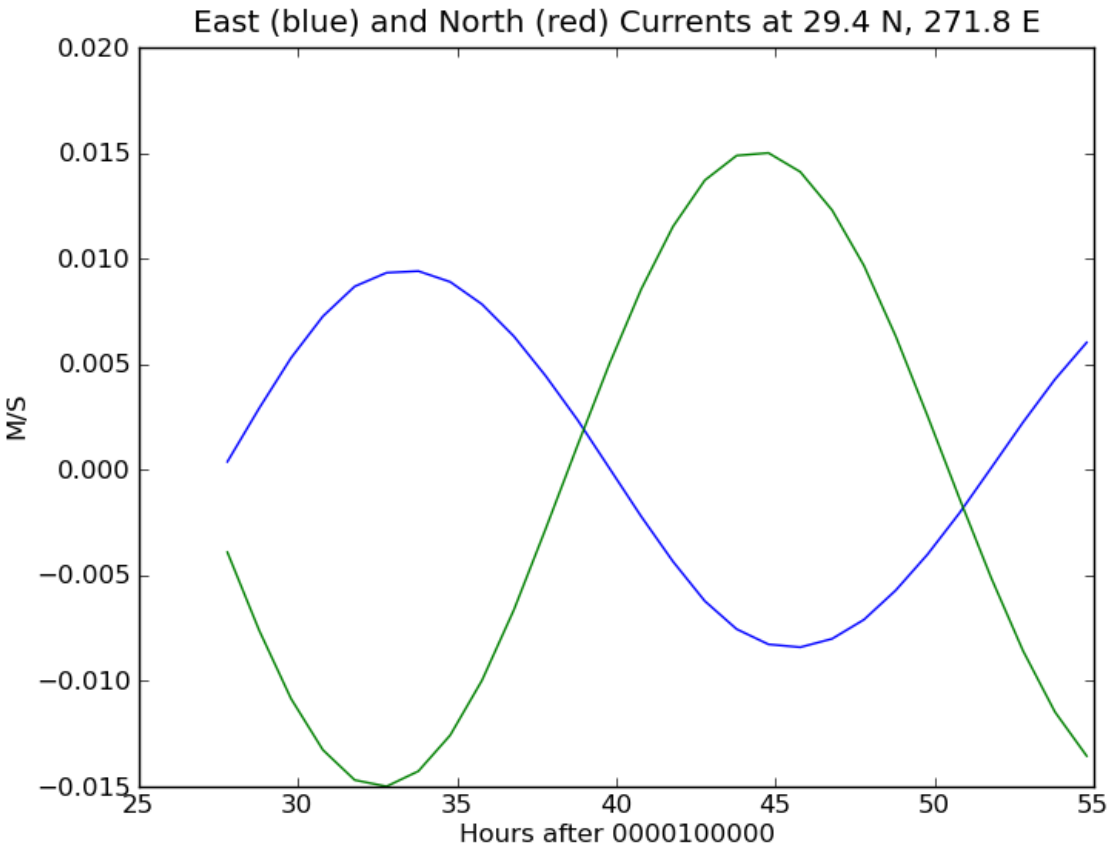


Figure 4.8. Plot from the MSB region made with the Matplotlib utilities.

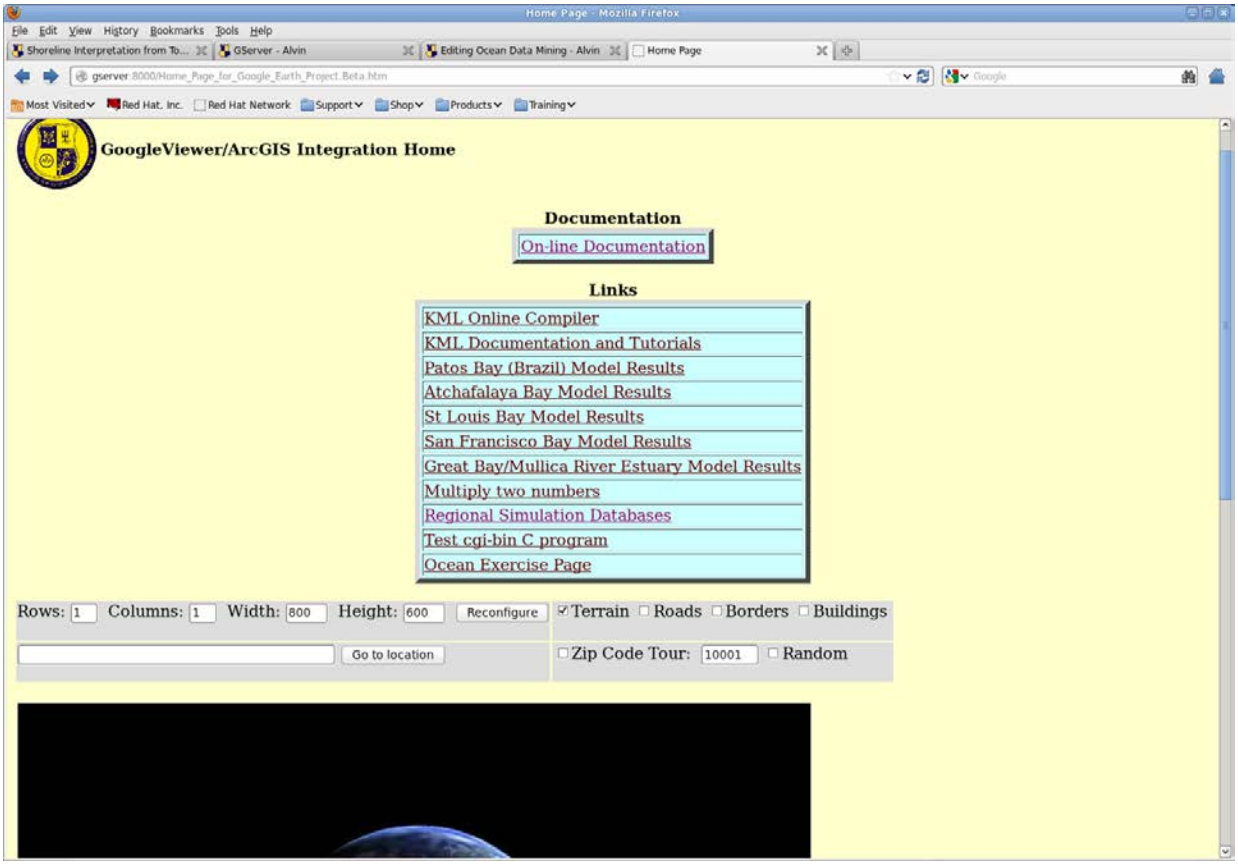


Figure 4.9. Opening page for the GServer:8000 page in the Oceanography division. The database portal is through the *Regional Simulation Databases* button.

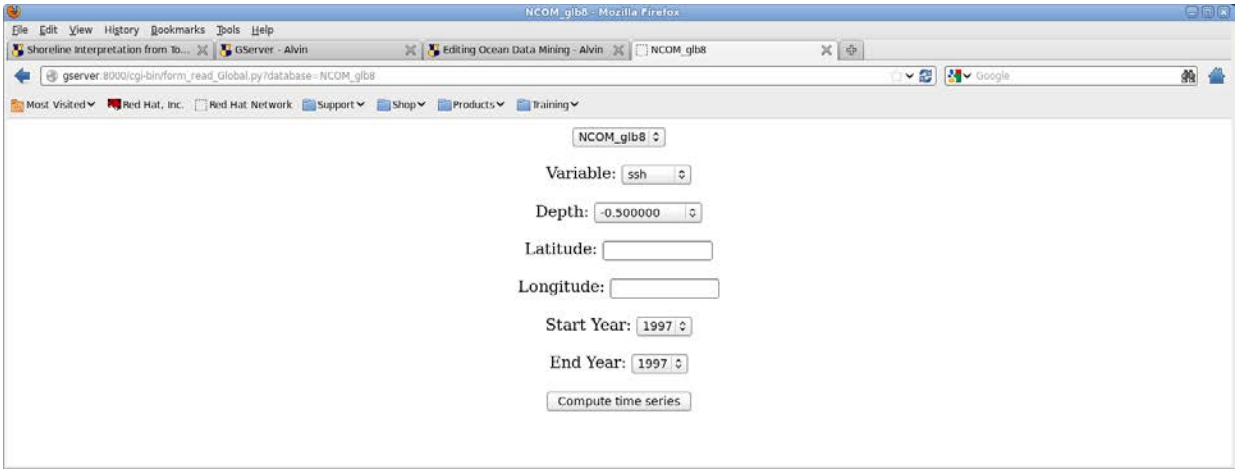


Figure 4.10. Form for reading specific variables and dates from the NCOM global database.

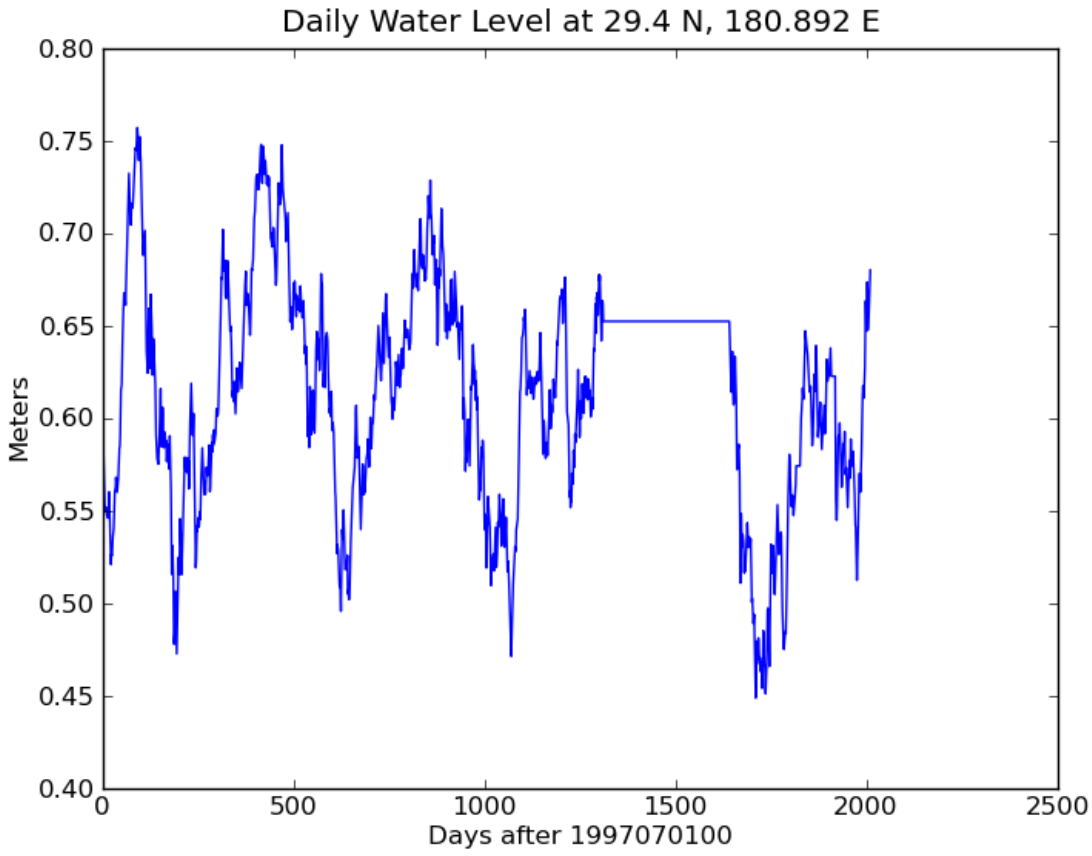


Figure 4.11. NCOM global model output at 24-hr intervals for 1997 to 2002 at 180.892° E, 29.4° N.

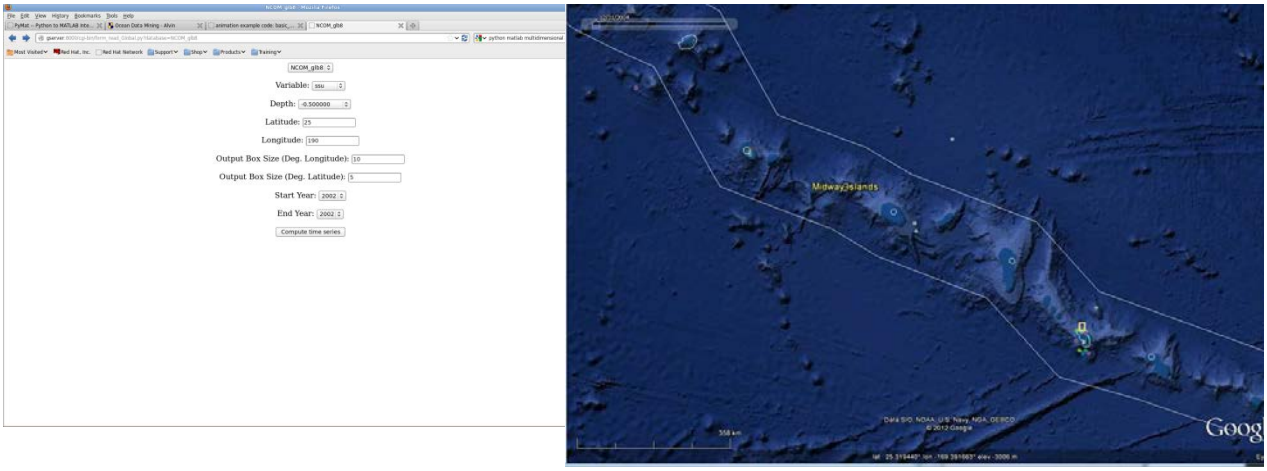


Figure 4.12. A. Menu for the global NCOM results. The *Output Box Size (Deg.):* box is used to extract a subarea from the entire database extent. **B.** Image of the N. Pacific Ocean as described in the menu from A. The islands are Midway Islands, NW of Hawaii.

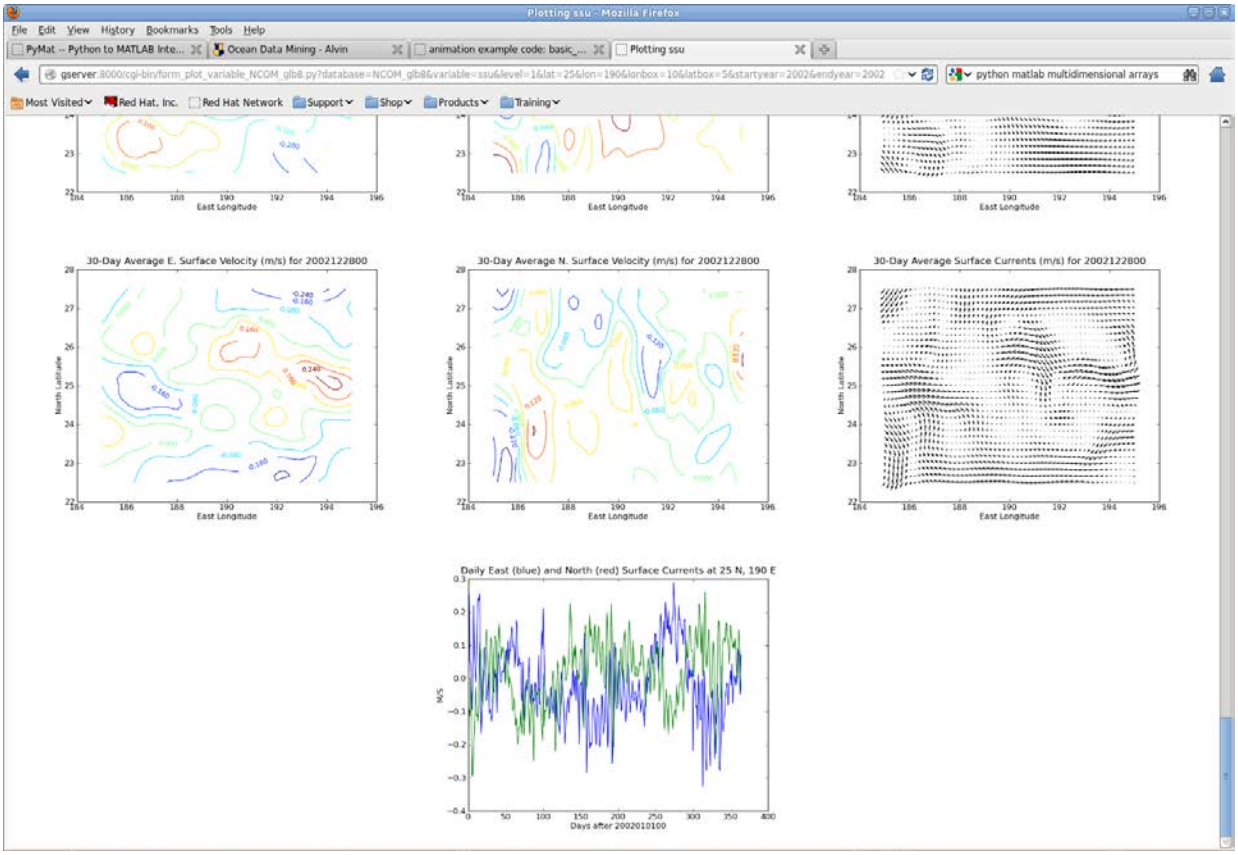


Figure 4.13. Browser result for December average surface currents and time series at the center of the output box.

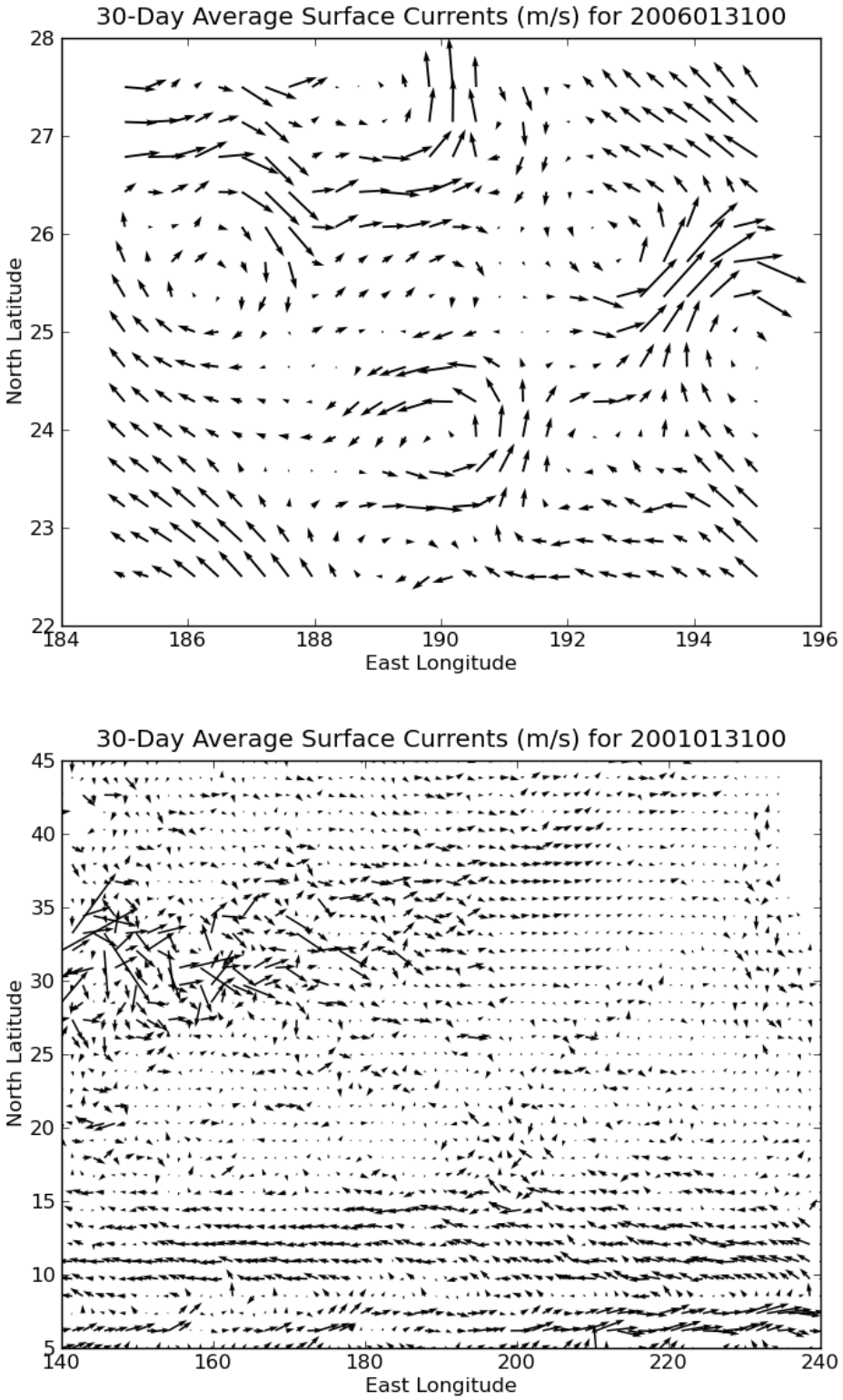


Figure 4.14. A. Jan 2006 average currents subsampled to 1/4°. **B.** Jan 2001 average currents subsampled to 1° for the N. Pacific.

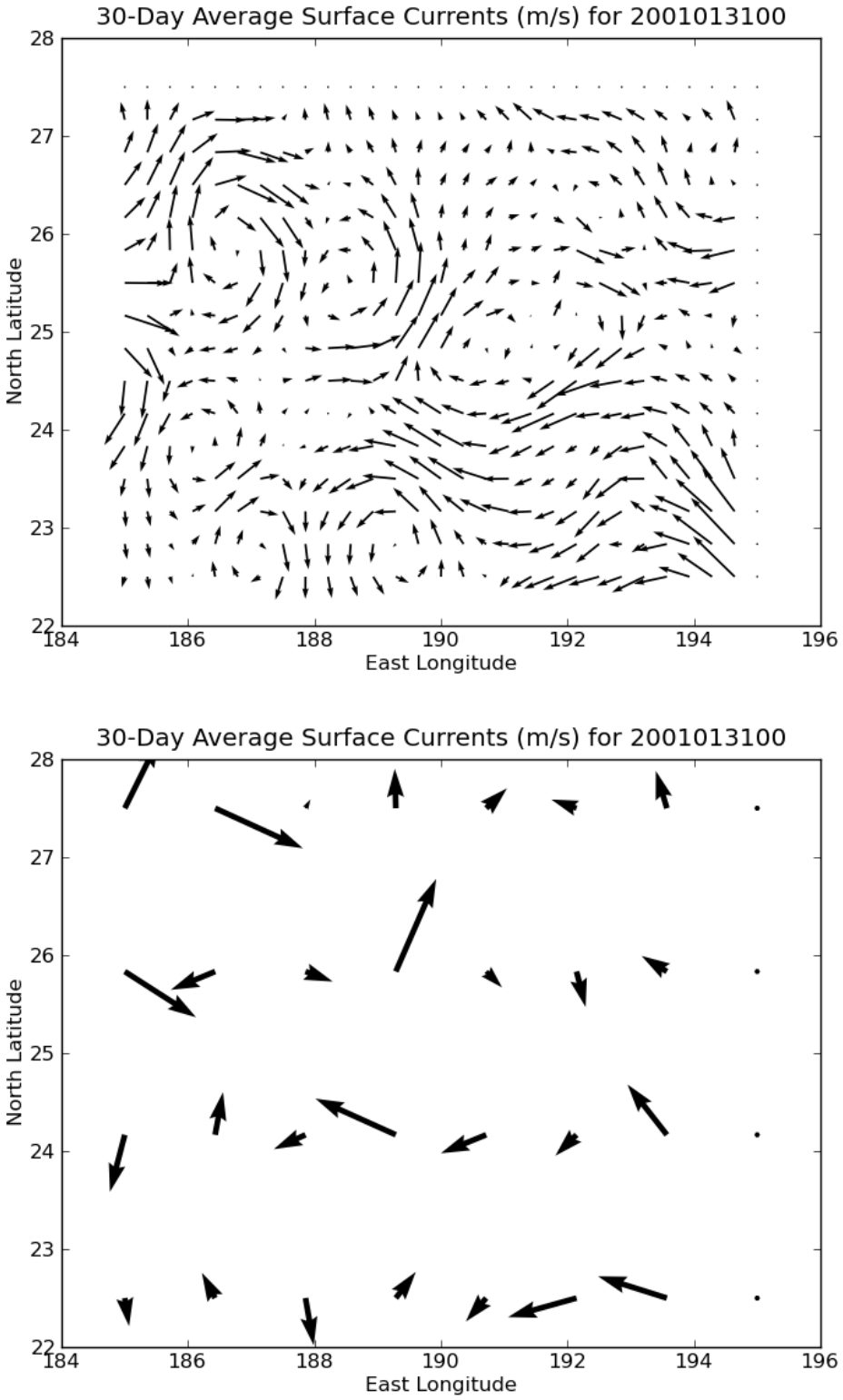


Figure 4.15. A. Jan 2006 average currents subsampled to 1/4°. **B.** Jan 2001 average currents subsampled to 1° for the N. Pacific.

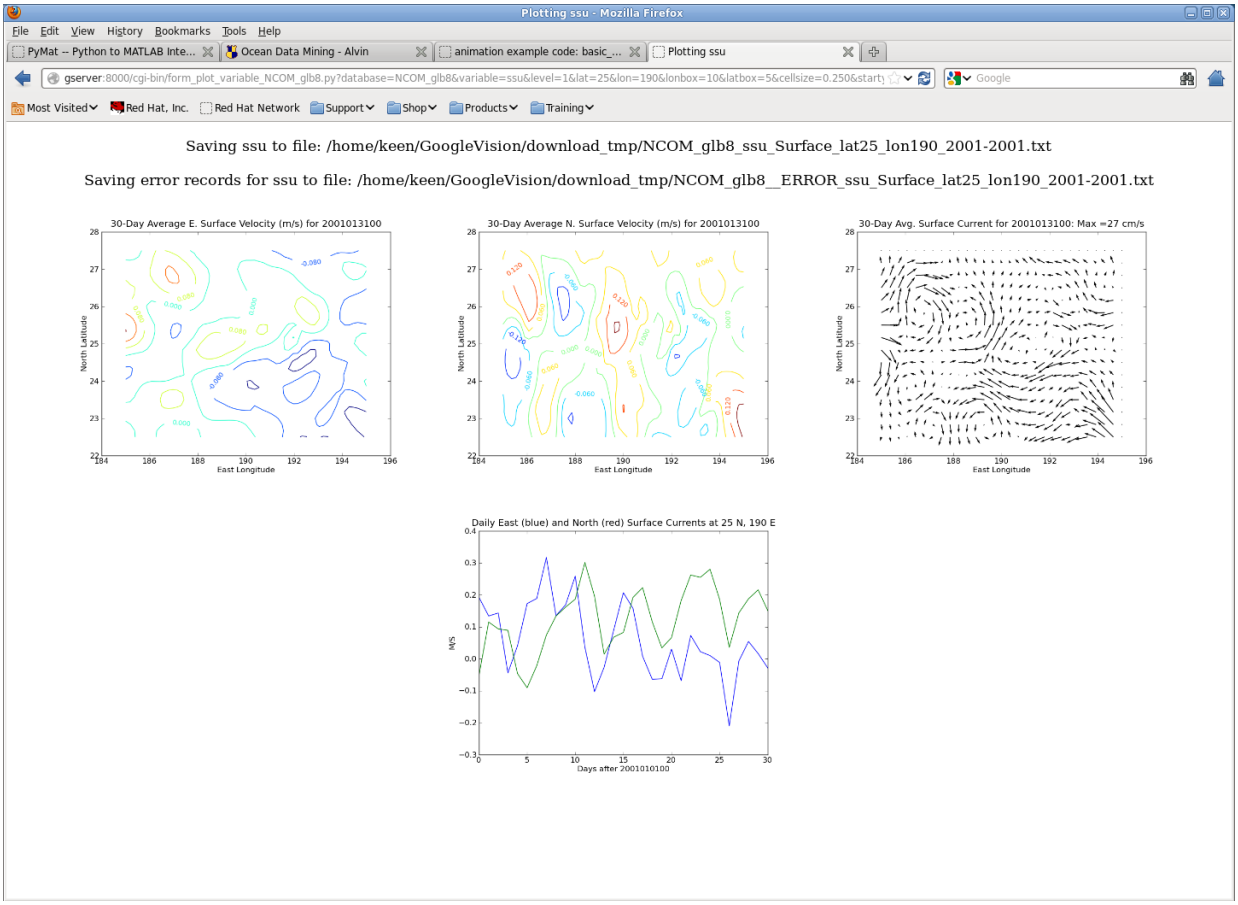


Figure 4.16. Example of output from the NCOM global 1/8° database.

5. Component Integration: Ocean Drifter Web Services

Introduction

A Lagrangian particle-tracking algorithm was also applied to potential contaminant transport within San Francisco Bay and Mississippi Sound (Keen and Holland, 2010). This model is useful for near-field studies like that at Hunters Point, San Francisco (Figure 5.2A) as well as longer transport paths within the full estuary (Figure 5.2B). The aqua particles have a settling speed of 10^{-3} mm/s, which is representative of fine clay flocs and particles. The yellow particles have a settling speed of 10^{-2} mm/s, which is more typical for large flocs and silt. The bathymetry is represented by shading. The vectors are the surface currents calculated by NCOM on the 30-m grid. Current speed (m/s) is indicated by shading as shown on the colorbar.

Drifter Model

The original drifter model is written in Fortran; the code was originally part of the ECOM (Estuary and Coastal Model) modeling system. The tracer and particle code, in addition to the sedimentation algorithm, were removed and formed into the HQCM (HydroQual Contaminant Model) system that includes the particle tracking model. This HQCM module was used for both Eulerian and Lagrangian simulations in several coastal transport studies (Keen and Holland, 2010).

DESCRIPTION

The current work is further decomposing this model into a particle tracking module. This software is being developed from previous studies at NRL (Keen et al., 2006; Keen and Holland, 2011). A commercial particle-tracking model was used by Keen et al. (2006). The particle tracking model uses three-dimensional current fields (*u, v, w*) and Gaussian random-walk dispersion [Lu et al., 2002]. A particle's location (*x, y, z*) after a time step Δt is given by:

$$(1)\quad x_{t+1} = x_t + (u \cdot \Delta t) + (\Delta D_{LT} \cdot u/s) + \Delta D_{H0}$$

$$(2)\quad y_{t+1} = y_t + (v \cdot \Delta t) + (\Delta D_{LT} \cdot v/s) + \Delta D_{H0}$$

$$(3)\quad z_{t+1} = z_t + (w \cdot \Delta t) + \Delta D_{V0}$$

where: u = x-directed velocity; ΔD_{LT} = horizontal dispersion; s = current speed; v = y-directed velocity; ΔD_{H0} = neutral horizontal dispersion; ΔD_{V0} = neutral vertical dispersion; w = vertical velocity. The dispersion terms are given by:

$$(4)\quad \Delta D_{LT} = (6 \cdot K_{D1} \cdot \Delta t)^{1/2} \cdot RAND_{-1}^{1}$$

$$(5)\quad \Delta D_{H0} = (6 \cdot K_{D2} \cdot \Delta t)^{1/2} \cdot RAND_{-1}^{1}$$

$$(6)\quad \Delta D_{V0} = (6 \cdot K_{DV} \cdot \Delta t)^{1/2} \cdot RAND_{-1}^{1}$$

where K_{D1}, K_{D2}, and K_{DV} are dispersion coefficients with values of 0.002, 0.002, and 0.0002 m^2/s, respectively. The dispersion terms are randomized for fractional values from -1 to 1 using the $RAND_{-1}^{1}$ term, which has a Gaussian distribution. Particles are introduced at specified points using a fixed production rate of 2 particles per day. As a particle settles through the water column, equations (1) to (6) are solved at its new level before the settling velocity is

applied. When the particle reaches the bottom it is deposited and cannot be resuspended. This method was applied for a year of archive results for the Papua New Guinea area (Figure 5.1). xamining particles in *Groups* 63 through 91 released at this location.

The model represented by equations (1) through (6) has been modified for a web-based application as described herein. Particle movement within the model coordinate system is accomplished using three terms: (a) advection by the 3D currents from a hydrodynamic model; (b) a random-walk component using the background mixing, *horcon*; and (c) sub-grid-scale diffusion based on the embedding flow, with *smagcon* and/or the Smargorinsky algorithm. The resulting terms are slightly different from Equations 1 through 6 above because the coordinates are in fractions of the local coordinate system, and the flow terms, *u/s* and *v/s* are not multiplied by a random coefficient, ΔD_{LT}. The modified equations are:

$$x_{t+1} = x_t + \Delta x_1 + \Delta x_2 + \Delta x_3$$

$$\Delta x_1 = \Delta t \cdot (u/h_x)$$

$$\Delta x_2 = \Delta D_{LT} \cdot [2 \cdot \Delta t (K_c/Pr_h)/ h_x^2]^{1/2}$$

$$\Delta x_3 = \Delta t \times \Delta_{x,y}[(h_y/h_x) \cdot A_h \cdot D]/(Pr_h h_y h_x D)$$

where: $A_h = K_h \cdot h_x h_y \times [\Delta_x u^2 + \Delta_y v^2 + 0.5 \cdot (0.25 \cdot \Delta_y u + 0.25 \cdot \Delta_x v / \Delta x)^2]^{1/2}$

Δt = time step

h_x, h_y = local cell dimensions along x and y axes

K_c = constant horizontal eddy diffusivity

K_h = dimensionless coefficient used in Smagorinsky diffusion term, A_h

Pr_h = horizontal Prandtl number

$\Delta_{x,y}$ = spatial gradient along both x and y axes

D = local water depth

The model structure can be defined using classes, which are contained in the following class files:

time.f:	time_ymdhm_to_1980minutes; time_1980minutes_to_ymdhm
utils.f:	gasdev; ran2; utils_smag
particle.F:	particle_main; particle_position; particle_position_write; particle_position_read
grid.F:	grid_new_ascii; grid_read_ascii; grid_read_nc; grid_new_vertical; grid_corners
io.F:	io_read_nc_fluxes; io_read_ascii_ssu; io_strlenf77; io_printint; io_getreal; io_getint
transform.F:	transform_point_bilinear_u; transform_point_bilinear_v; transform_point_bilinear_w; transform_point_bilinear_h; transform_point_bilinear_aam; transform_point_bilinear_kh; transform_point_bilinear_ud; transform_point_bilinear_vd; transform_point_idw_init; transform_point_idw_interpolate

The following files are also used:

Header file: hqcm-TEST.inc, hqcm-glb8.inc

Build files: HqParticle.pkg, Makefile

Fortran 90 source code: nf90_handle_err.f90

Executable: HqParticle_Linux.ex

The last is the executable file. There are no parameter input files; these values are defined as parameters in the header file, **hqcm-glb8.inc**.

restart: if true, a file will be read to continue a previous simulation with continuous dates but not time steps

dtgstart: WHOI format start date for simulation series (e.g., 200503150000)

dtgrestart: WHOI format restart date (e.g., 200503150000)

dtgstop: WHOI format stop date for current run (e.g., 200503150000)

logunitp: flag for writing a log file (not equal to 6, name = runlog-YYYYMMDDHHMM.txt)

hydfreq: frequency in hours to read a new set of forcing files

printfreq: frequency to update the particle location files for each source (hours)

dti: time step in seconds

horzmix: *CONSTANT* = use the value of *horcon*

SMAGORINSK = calculate using Smagorinsky (1964) algorithm

vertmix: set to *CONSTANT* to use a value of *khcon* instead of reading from a file

vertflux: set to *CONSTANT* to have no vertical transport of particles instead of reading from a file

horcon: constant value of horizontal mixing (m^2/s)

smagcon: coefficient used in Smagorinsky horizontal mixing (dimensionless)

hprnu: ratio of particle mixing to the horizontal mixing by the background flow

khcon: constant value of vertical mixing coefficient to use (m^2/s)

vprnu: ratio of particle mixing to the vertical mixing of momentum (*kh*)

cmax: maximum value of concentration after conversion from particles (not implemented)

relhr: frequency of particle release (hours)

npart: number of particles per release

nconv: total number of releases before conversion of particles to concentration (not implemented)

(note: total no. of particles in system = *npart* × *nconv*)

irelsthr:	beginning hour of particle release
npclass:	number of particle classes
nsource:	no. of sources of particles
epf:	error/message print frequency in seconds
srclon:	array of longitudes for particle sources
srclat:	array of latitudes for particle sources
srcname:	array of names of particle sources (max length = 10 characters)

The horizontal mixing processes are controlled by model variable *horzmix*. The background mixing is always calculated; if it is not desired, *horcon* must be set to zero. If *horzmix* = *SMAGORINSK* or *MODEL*, the flow-dependent term is either calculated or read from files, respectively. Thus, setting *horzmix* equal to *CONSTANT* will turn these off no matter what value of *smagcon* is set in the input file (**particle.in**).

The output consists of a file for each source location. This text file contains the following columns: (1) longitude; (2) latitude; (3) depth below surface; (4) model day for current output; (5) source number; (6) particle group number (i.e., time since beginning of simulation); (7) particle number within group.

The original model used the concept of a group to represent the *age* of particles. For example, all of the particles released from all sources are assigned to group 1, and *older* groups of particles are increased to the next higher group. Thus, the oldest particles have the largest group number. This is a way of dealing with the age of particles that chemically react with the environment. However, it is not possible to plot individual particle tracks without special output. The current version replaces this algorithm by assigning the group number to all newly created particles. This means that the oldest particles are in group 1. This allows paths to be tracked more easily.

OPERATION

This is an overview of the model operation. This description consists of three parts: (1) the coordinate system, or reference frame; (2) particle tracking; and (3) particle movement. A good understanding of part (2) is important to maximize the usefulness of the results. The computation of (3) is the area that will probably require additional work in the future because of differences in grids and forcing fields. This discussion omits the forcing fields because they are nonstandard and special software is required for new applications.

The model operates in a general curvilinear coordinate system for the horizontal plane. The input/output indices are in longitude and latitude. These are converted to the corners of the axes by subroutine *grid_corners*. This function converts the georeferenced coordinates to meters from index (0,0) in the curvilinear coordinates. It must be modified for applications other than a Cartesian grid, which is its current configuration. Particle positions within this model coordinate system are transformed to longitude and latitude for output using *xcor* and *ycor*,

which are not used for tracking particles during a simulation. Instead, individual particles are tracked using arrays, *xp*, *yp*, and *zp* for the *x*, *y*, and *z* axes. These locations are real number that are fractions of cell (*i*, *j*). This is done in a dead-reckoning fashion with each new position calculated using the forcing fields from the last position, starting with the location of its source. The vertical coordinate system is based on *z* levels at fixed depths (meters negative downward) that decrease down: e.g., -1, -2,...-10, which are converted to *zp*.

Every particle released during a simulation is tracked individually and unique identifiers are written to the output files. There is one output file for each source (e.g., PARTICLE_FPP_201101010000.TXT). The date in the file name is the input parameter *dtgstart*, or *dtgrestart* if the current simulation is restarted from a previous run. Each particle is identified by its source, group, and particle number. The sources are input from the parameter list. A *group* of particles is released at frequency *relhr* and remains tagged as a common group. The current mode has been modified from the original as follows. As particles were released, they became members of group 1, which therefore always contained the newest particles. This is difficult to use to track individual particles, however, because a particle's group changes every release. This has been avoided by leaving particles in the group to which they are assigned upon release. Thus, group 1 contains the oldest particles and individual particles can be tracked throughout any length simulation. Any number of particles (up to *npartm*) can be released every time. This is useful to evaluate the impact of randomness at sub-grid-scales is described in the next paragraph. This description of particle properties allows any combination or release times to be identified and plotted. For example, the simulation of particles released immediately following the nuclear accident at Fukushima Power Plant Number 1 between 11 March and 1 April, 2011, can be simulated using a run that starts on 1 January 2011 by examining groups 63 through 91.

The new particle positions are calculated in the main program, *particle_main*. The random coefficients, ΔD_{LT}, are calculated by functions, *utils_gasdev* and *ran2*. Subroutines *transform_point_bilinear_ud*, etc., interpolate the hydrodynamic fields to the current particle position, and the Smagorinsky coefficient field A_h is calculated for each new current field by subroutine *utils_smag*. Analogous equations are used for the *y* and *z* axes, except with vertical mixing constant and diffusion coefficients in variables *khcon* and *vprnu*, respectively.

The drifter model is fast but it is convenient to have the capability of initializing the particle locations in the case of an accidental stoppage or to evaluate the tracks. This has been implemented using a couple of added functions. The input parameters (contained in file, **particle.in**) now include a date-time group for the simulation to be used as a restart source (*dtgstart*) as well as an added variable for the date-time to restart (*dtgrestart*). The *dtgstart* variable is used as part of the particle file (e.g., **PARTICLE_ship_01_200603090000.TXT**) used to initialize all existing particles up to the day after the initial date-time, which is transformed into a day since *dtgstart*. These longitudes and latitudes are transformed into grid coordinates (i.e., 1.0, 1.001, etc) by subroutine *read_particle* and passed to the calling program along with the source, group, and particle numbers associated with each line in the file. These values serve as the first locations for the continuing computations. Note that the run length in the header file refers to the entire simulation series *including the previous simulations* from *dtgstart*.

The restart capability has been tested on the NPAC domain (Figure 5.3). A drifter simulation was started on 9 March 2006 00:00 (200603090000) and run for 174.5 days. These results are discussed as part of Case 2 below. This simulation was restarted on 30 August 2006 00:00 and continued until day 297.5, at which time the model ran out of forcing fields from the NCOM 1/8° database. The surface currents at station *ship 01* for this interval (Figure 5.4) show a daily signal as well as a 90 day cycle in eastward currents. The northward current is also weakest during this time.

The resultant distribution of all particles (*npart* = 2) and groups (i.e., daily; *relhr* = 24) at the end of the simulation (Figure 5.5A) shows a lot of spreading in the western Pacific near the Kurishio current area. This plot includes particle locations from both particle files: **PARTICLE_ship_02_200603090000.TXT** and **PARTICLE_ship_02_200608300000.TXT**. The plotted date thus only includes particles from the latter file. A track from particle 1 in group 1 (the first group released) (Figure 5.5B) shows results from both files, however. The blue line is the newer particles (through 30 Aug 2006) and the red is the location of the end of the year.

Analysis Methods

Supporting software for this work consists of two types: (1) processing raw files into formats suitable for use in the model and postprocessing by software like Matlab; and (2) scripts and programs used for visualization and graphics

DATA PROCESSING

The North Pacific Trash Vortex application uses the data from NCOM 1/8° database, which are stored in NetCDF files. These are processed into **csv** files using the *cgi-bin* script *form_plot_variable_NCOM_glb8.py*. These files are placed in directories after being processed by a shell script *rewrite_csv.sh*, which replaces dashes with "-9999". The resulting files are prepended with "new_" to distinguish them from the originals. They are readable by all of the programs in use for this project. The raw output (**csv**) files are in the spatial and temporal subsample frequencies defined in the extraction Python script; currently these are 1/4° and 24 hours, respectively.

The original drifter model does not interpolate any forcing fields. It uses each time-stamped field until the next is required and it runs on the grid file supplied with the **csv** files (e.g., **GRID_0284x0138.TXT**). This has been modified for using monthly mean fields to interpolate in time. The *raw* files are averaged in time (30 days) using a Matlab script, *time_space_average.m*. This script produces files called

```
mean_Jan2001_ssu.csv->   new_NCOM_glb8_ssu_Surface_lat25.0_lon190.0_box=100-
40_2001011500_cell0.25.csv@
```

These are linked to files of the old naming convention to avoid special cases. However, the file dates that are available are listed in a text file, **time.in**. This is used to read the hydrodynamic field dates. The header row has been removed in order for the Fortran program *io_read_ascii_ssu* (file **io.F**) to read them the same was as the *raw* **csv** files. The spatial interpolation is completed by the transformation subroutine as is currently done, so no further processing for spatial simulations is required.

VISUALIZATION AND GRAPHICS

The results from the model are processed with Matlab scripts:

```
/home/keen/PROJECTS/Drifter_Studies/Software/test_bathy_01.m
```

is used to process Case 1. The results from the North Pacific using the global NCOM 1/8° results are plotted with:

```
/home/keen/PROJECTS/Drifter_Studies/Software/npac_glb8.m
/home/keen/PROJECTS/Drifter_Studies/Software/plot_vectors_npac.m
```

Script, *npac_glb8.m* currently produces 10 figures.

- Figure 1 is the location of six particle sourcess plotted over bathymetry.
- Figure 2 is a plot of all particles/groups from the simulation. If a particle was ever at a location, it is plotted.
- Figure 3 plots the path of a specified particle/group.
- Figure 4 is a time series plot of east and north currents at six particle source locations (e.g., **YH-HI**). These are interpolated directly from the global database by the *Python* script *form_plot_variable_NCOM_glb8.py* when it extracts the currents. They are read from a text file (e.g., **FORCING_ship_01_200608030000**) with space-delimited fields, *time*, *u*, and *v*.
- Figures 6-11 are time series plots of the dimensional forcing terms used to calculate a particle's new location.

The script, *plot_vectors_npac.m*, is used to plot a cut-out of the subsampled vectors from the global database. The data are read from files (e.g., **new_NCOM_glb8_ssu_Surface_lat25.0_lon190.0_box=100-40_201103110000_cell0.25.csv**, which are created from the original csv files from the *Python* script, *form_plot_variable_NCOM_glb8.py*. The original files contain "--" where NODATA values existed in the original NetCDF files. These could not be read by Matlab, so a shell script, *rewrite_csv.sh*, replaces these bits with “-9999”, which appears to work fine in Matlab. This Matlab script plots surface currents over contours of the divergence of the surface currents. The divergence is calculated by the Matlab function, *div*.

STATISTICAL METHODS

The particle distributions are analyzed in two ways: (1) normalized cumulative Lagrangian separation (Liu and Weisberg, 2011); and (2) cumulative (time/space) particle distributions, which can be used to calculate PDF's (LaCasce, 2008). These are calculated using the M0T0S0 experiment from 2001 as a base run. The separation distance is calculated between the same particle from the same group in paired analyses. The separation distance d_n is used to calculate a normalized separation as shown in Figure 5.6. From this index, a trajectory skill score is defined. This index is calculated by the Matlab script *separation_index.m*.

Model Verification

This section demonstrates that the calculations are consistent with the equations and that the basic functionality is robust.

CASE 1: IDEALIZED DOMAIN WITH TIDES AND VARIABLE WIND

A standard test domain (Figure 5.7) consists of a N-S oriented straight shoreline on the west with a uniformly dipping bottom. The forcing consists of semidiurnal tides and a weak southerly wind. Particles originate at grid cell (3, 15, 1), where the indices refer to the *x*, *y*, and *z* axes, respectively. The forcing is supplied by flux files from NCOM, which are the transports (m^3/s) in the east (Figure 5.8A) and north (Figure 5.8B) directions. These are converted by the drifter model into velocities. The available fields are interpolated in time as required.

The results of the drifter computations are output as a series of longitudes and latitudes for each particle. The first example uses one release time with only one particle; the results (Figure 3) are thus readily interpreted as being the subsequent location of one particle. The resulting output also includes only one group. All of the particle locations reported for 24 hours from this simulation demonstrate the dominance of tidal flow (Figure 5.9A). Note that the entire range of positions only spans 10 m in the *x* direction and 8 m parallel to the *y* axis.

The trajectory reflects two processes: (1) advection by tide and wind flow; (2) mixing from the model output, including a random component. This example neglects (2), however, but the interaction of tides and wind flow is apparent. These points can be joined with a line to form a path. This is a potential alternate output method for lower particle numbers like this (Figure 3B). Figure 5.10A shows the same path for group 1 as Figure 5.3B but the addition of another group produced at 12 hr has no impact on the first group's path. Note, also, that the second group does not plot at the origin because the output time does not exactly coincide with the release time.

These simulations do not use horizontal (*horcon* = 0) or vertical (*khcon* = 0) mixing in order to reproduce the path of group 1. When a vertical mixing coefficient of *khcon* = 0.0001 is used (Figure 5.10B), a different result is produced because the transport/velocity field is three dimensional and the particle is randomly moving between layers. The random component of process (2) produces different results for the number of particles because an index *dum* is used to generate the random component from subroutine *gasdev*. However, this is only pseudo-random because the results are reproduced for simulations with the same number of particles.

The final simulation is intended to demonstrate the overall performance of the model. This run is for 5 days, with 3 particles released from the source every 12 hours. Both vertical and horizontal mixing were turned on (*khcon* = *horcon* = 0.0001). The resulting distribution (ignoring depth) at day 5 (Figure 5.11) shows the overall northward drift of the coastal flow.

CASE 2: NORTH PACIFIC FROM GLOBAL NCOM

This is the preliminary validation study for the Pacific Trash Vortex study. The domain (Figure 5.3) extends across the North Pacific, centered on the Hawaiian Islands. The forcing consists of daily surface currents from the Global 1/8° NCOM archive. Particles originate at grid cell (140, 25, 1), where the indices refer to the *x*, *y*, and *z* axes, respectively. The forcing is two-

dimensional only and the drifter model has been modified to run this particular case with a compiler option (-DRUN_GLB8). This simulation is 30 days in duration.

The time step is 6 hours. The forcing is supplied by unstructured current files from NCOM, which are interpolated in space only. The time interpolation needs to be done later, but it is not critical for the large scale of this problem. The general trend can be seen in Figure 5.12; the flow is predominantly westward with a weakening in the middle of March.

We can complete an analysis of this simulation as for Case 1. The particles are released every 24 hours and their location printed at 12 hr intervals. There are thus 30 groups, with group 1 being 30 days old. A plot of groups 1 and 15 (Figure 5.13) shows the steady westward transport of surface particles by the north equatorial current (see below). The blockiness of the paths is an artifact of the plotting function, *prtpart*, that should be improved.

The final verification of the particle model is a 180 day simulation using 2 particles per release. The distribution of particles after 160 days (Figure 5.14A) clearly follows the North Equatorial current to the west with a scatter of particles at the western extreme of their tracks. The tracks for individual particles released at 0 and 15 days (group 15) shows that they have been caught by an eddy near 160° (Figure 5.14B). We can see that the younger particle (blue line) overshot but then was trapped on the western side of the eddy and begins to follow the same path as the older one.

The simulation stopped on August 31 because the particle step exceeded the grid size, which is ~35 km. The normalized score has the advantage of negating the bias in error for regions with strong flows, like the Kuroshio Current. The error in particle location (Figure 5.15A) indicates large differences when the particle originating from the shipping lane between Yokohama and Hawaii is influenced by these currents. This particle took a NE track (Figure 5.14B) before turning east in the KESS. The distances traveled by these matched particles (Figures 5.15B and C) are the very similar because they use the same forcing. The normalized separation distance (Figure 5.15D) reflects differences in travel time as well as error. There is a potential problem, however, if leg lengths are short. Extreme values of D/L_o can occur. We will handle these by assigning a maximum to the output. This is discussed by Liu and Weisberg (2011), who suggest a value of ~1 for these cases.

The total particles are saved to the files. These can be plotted as number of particles in each grid cell for all time steps. These are plotted as individual contours or as total plots (Figure 5.16) by Matlab script, *particle_pdf.m*. Differences between test experiments and the baseline are used in later work.

Some cells, especially those near dynamic sources, are reoccupied numerous times. These are counted only once for this metric, which is a measure of spreading. This can be transformed into an area by multiplying with the cell area $(0.35° \times 0.3°) \cdot 111.1172$ km. This is an estimate only since area is not conserved on the analysis grid. This parameter is calculated in Matlab script, *particle_pdf.m*.

Matlab script *particle_path.m* calculates the trajectory parameters for particle 1 from group 1. The output consists of the following statistical parameters for each source location: (1) cumulative surface area, A_c; (2) total path length for the first particle released, $L_{t,\,1}$ (km); (3) mean path length for the first particle released, $L_{m,\,1}$ (km); and (4) mean trajectory speed (m/s), $S_{m,\,1}$.

Visualization with ArcGIS

For the area covering much of the northern Pacific centered on Hawaii, addressing the Pacific Trash vortex, modeled surface currents were derived from a long-term set (one year) of NCOM fields in terms of daily averages. These fields are the bases for the modeled drifter results that are ingested in ArcGIS. The output from the drifter model consists of hundreds of thousands of records, each with longitude, latitude, depth, fraction of the day, source, group number, and particle number and file (Table 5.1). One group of two tracers up to 348 groups was released once a day throughout the year. The data records are ordered by day, and all released drifter locations are recorded every half a day. The group number represents the day minus one of release and was tracked for not more than 195 days. Thus, the two drifters from group 1 are among the oldest on the list, their final resting places at around 120.88 W and 27.065 N amongst a number of other early release drifters.

After converting the ASCII output into a CSV file, the data were added to the ArcGIS catalog (Add Data button) and incorporated as a table that is plotted on the map (Display XY Data…). More that 200,000 points are plotted in Figure 5.17, many of which are overlapping each other at specific nearest-grid locations, and not all of these points are apparent. Refreshing the map takes significant time. Nevertheless, this data processing step facilitated analysis of the output such as finding the oldest point on the list.

More information about the data can be displayed when we change the plotting attributes. In Figure 5.18 the group number is plotted by colour with a scale such that green indicates the most recent and red is the oldest. Naturally, we would have thought that the the points furthest away from the source (i.e. from San Francisco Bay) would be the oldest, but distances traversed by individual drifters vary widely. There are a significant number of red values plotted closer to the source indicating that many drifters did not move far. Realizing that we still have this issue with overlapping points as they are stuck to nearest grid locations (an aspect of the model output we cannot control here), the points we see are the last ones that arrived at that location. This indicates that points close to the source were not visited much more in the course of the model run. For example, according to an interrogation at a red point locatedn in the northern part of all the plotted points, the newest of 106 co-located points was released on day 60, and no other points arrived there since.

It would be convenient to plot the distribution of drifters but, had the model drifter program output values at more realistic locations, then we would be able to see a better distribution. A simple KMZ exporter tool was used in ARCOAS that resulted in the display in Figure 5.18. This is really a PNG file converted from the original features plotted in ArcMap, preserving the information about the age of the last arriving drifters.

Table

PARTICLE_SFB_200101010000.csv Events

Longitude	Latitude	Depth	Day	Source	Particle	Shape *	Group
235.598618	36.804371	-1	0	5	1	Point Z	1
235.598618	36.804371	-1	0	5	2	Point Z	1
235.598602	36.804371	-1	0.5	5	1	Point Z	1
235.598602	36.804371	-1	0.5	5	2	Point Z	1
235.598602	36.804371	-1	1	5	1	Point Z	1
235.598602	36.804371	-1	1	5	2	Point Z	1
235.598602	36.804344	-1	1.5	5	1	Point Z	1
235.598602	36.804344	-1	1.5	5	2	Point Z	1
235.598618	36.804371	-1	2	5	1	Point Z	1
235.598618	36.804371	-1	2	5	2	Point Z	1
235.598618	36.804344	-1	2.5	5	1	Point Z	1
235.598618	36.804344	-1	2.5	5	2	Point Z	1
235.598618	36.804344	-1	3	5	1	Point Z	1
235.598618	36.804344	-1	3	5	2	Point Z	1
235.598602	36.804344	-1	3.5	5	1	Point Z	1
235.598602	36.804344	-1	3.5	5	2	Point Z	1
235.598618	36.804359	-1	4	5	1	Point Z	1
235.598618	36.804359	-1	4	5	2	Point Z	1
235.598602	36.804344	-1	4.5	5	1	Point Z	1
235.598602	36.804344	-1	4.5	5	2	Point Z	1
235.598618	36.804371	-1	5	5	1	Point Z	1
235.598618	36.804371	-1	5	5	2	Point Z	1
235.598633	36.804371	-1	5.5	5	1	Point Z	1
235.598633	36.804371	-1	5.5	5	2	Point Z	1
235.598618	36.804359	-1	6	5	1	Point Z	1
235.598618	36.804359	-1	6	5	2	Point Z	1
235.598618	36.804371	-1	6.5	5	1	Point Z	1
235.598618	36.804371	-1	6.5	5	2	Point Z	1
235.598633	36.804371	-1	7	5	1	Point Z	1
235.598633	36.804371	-1	7	5	2	Point Z	1
235.598602	36.804371	-1	7.5	5	1	Point Z	1
235.598602	36.804371	-1	7.5	5	2	Point Z	1
235.598633	36.804371	-1	8	5	1	Point Z	1
235.598633	36.804371	-1	8	5	2	Point Z	1

1 (1 out of 2000 Selected)

PARTICLE_SFB_200101010000.csv Events

Table 5.1. ArcGIS table of values resulting from the drifter model whose source was out of the mouth of the San Francisco Bay.

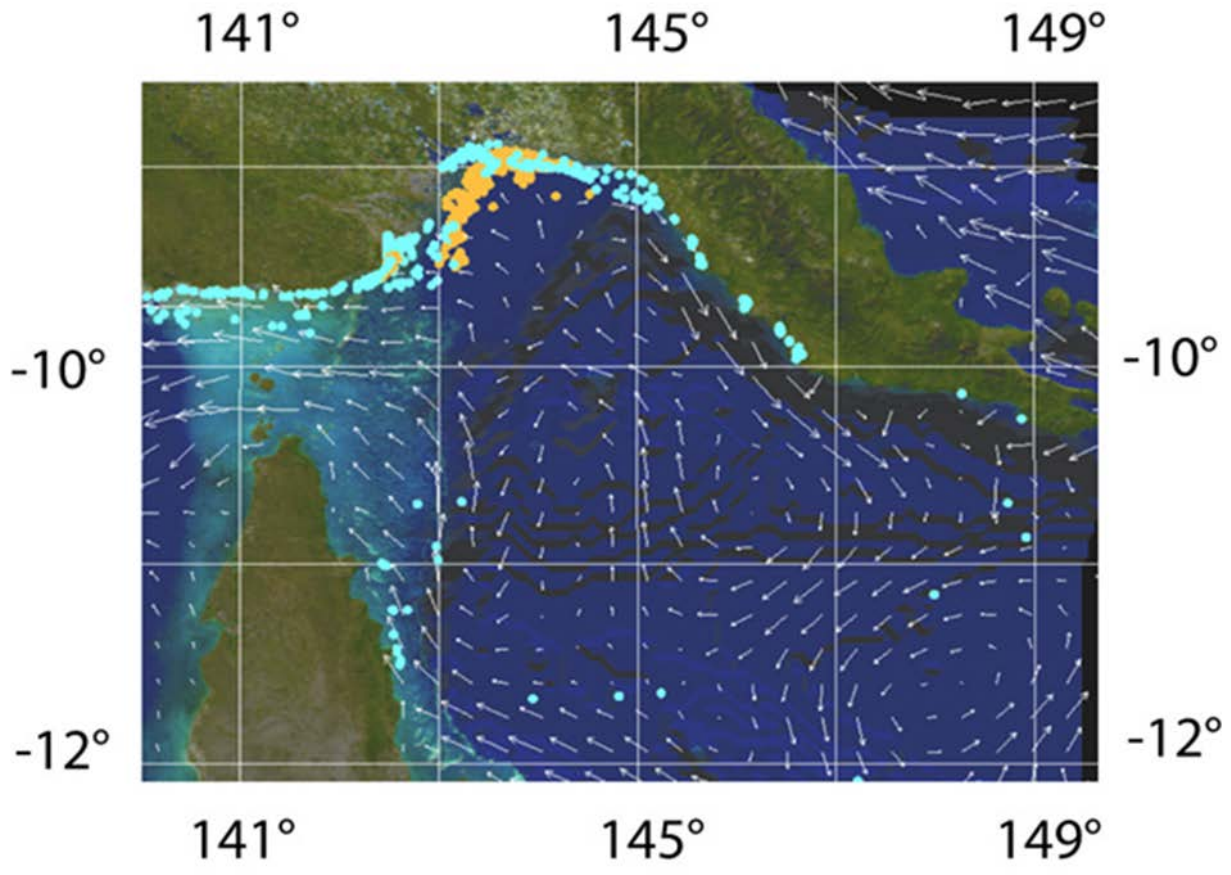


Figure 5.1. The distribution of particles after one year of integration. Topography is overlaid on model results using MODIS imagery. The yellow particles represent clay minerals and the blue represent finer particles (Keen et al., 2006).

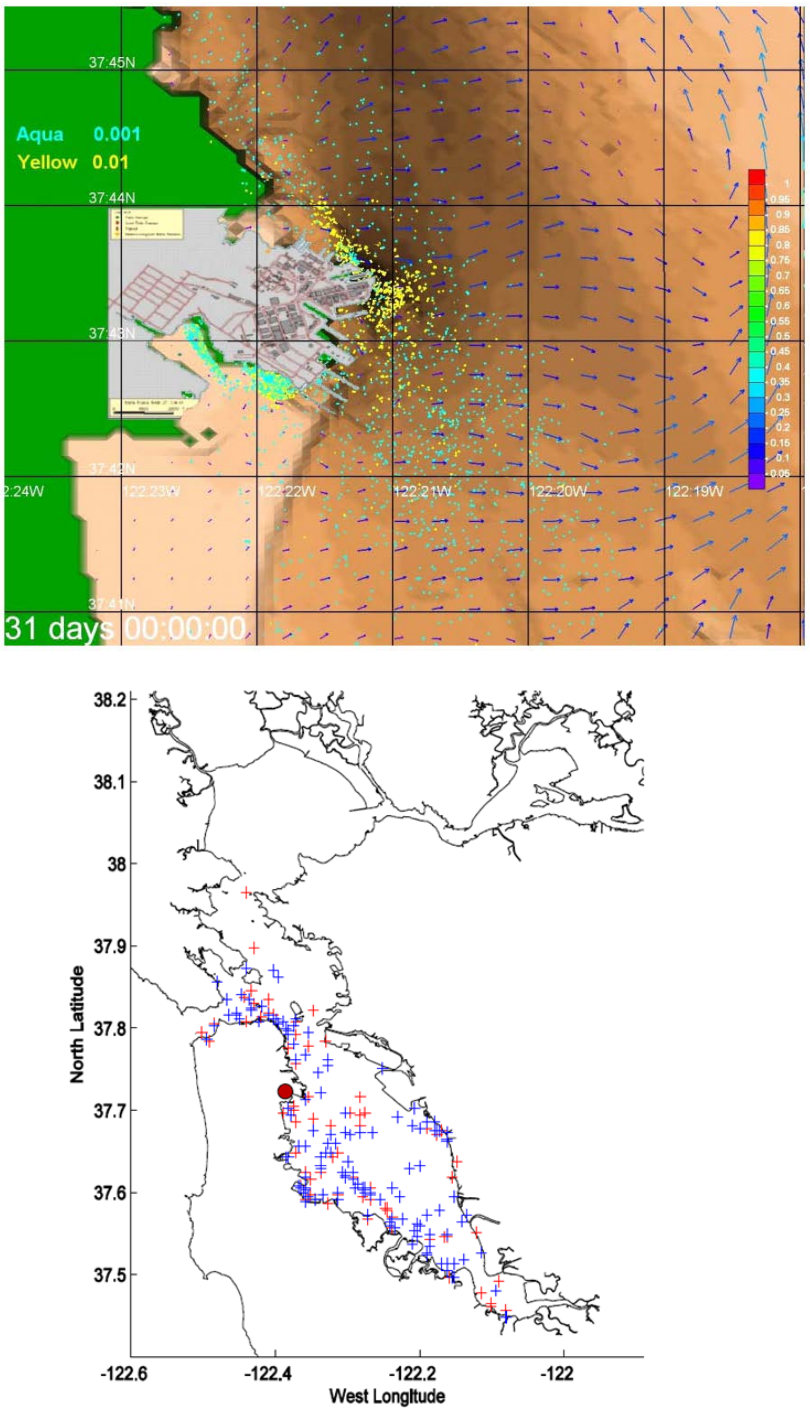


Figure 5.2. Particle tracking results for San Francisco Bay. **A.** Particle distributions near Hunters Point computed by a particle tracking algorithm for Jan. 2004. Particles were released at the locations indicated by the circles. **B.** Particle distribution in San Francisco Bay after 15 days (red pluses) and 28 days (blue pluses) for Feb. 2004. The hydrodynamic forcing is from the NCOM SFB grid (300 m). One particle was released every 24 hr just south of HPS (red circle).

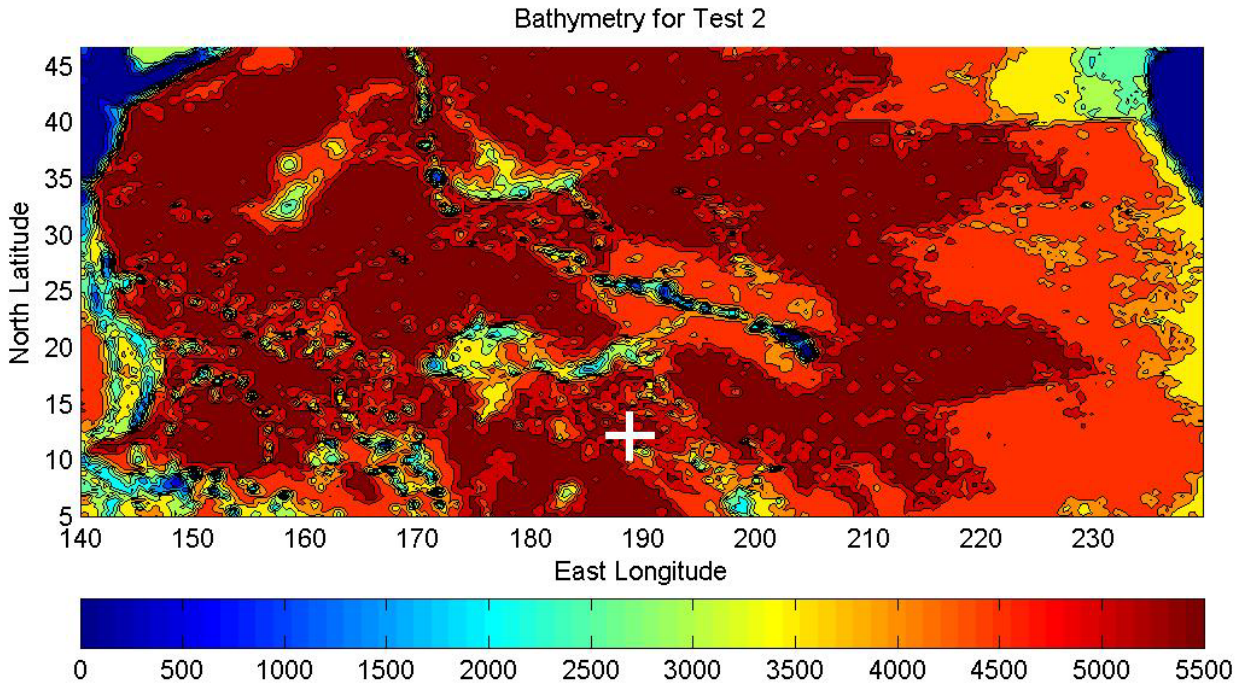


Figure 5.3. North Pacific bathymetry. The particle source is indicated by the white +.

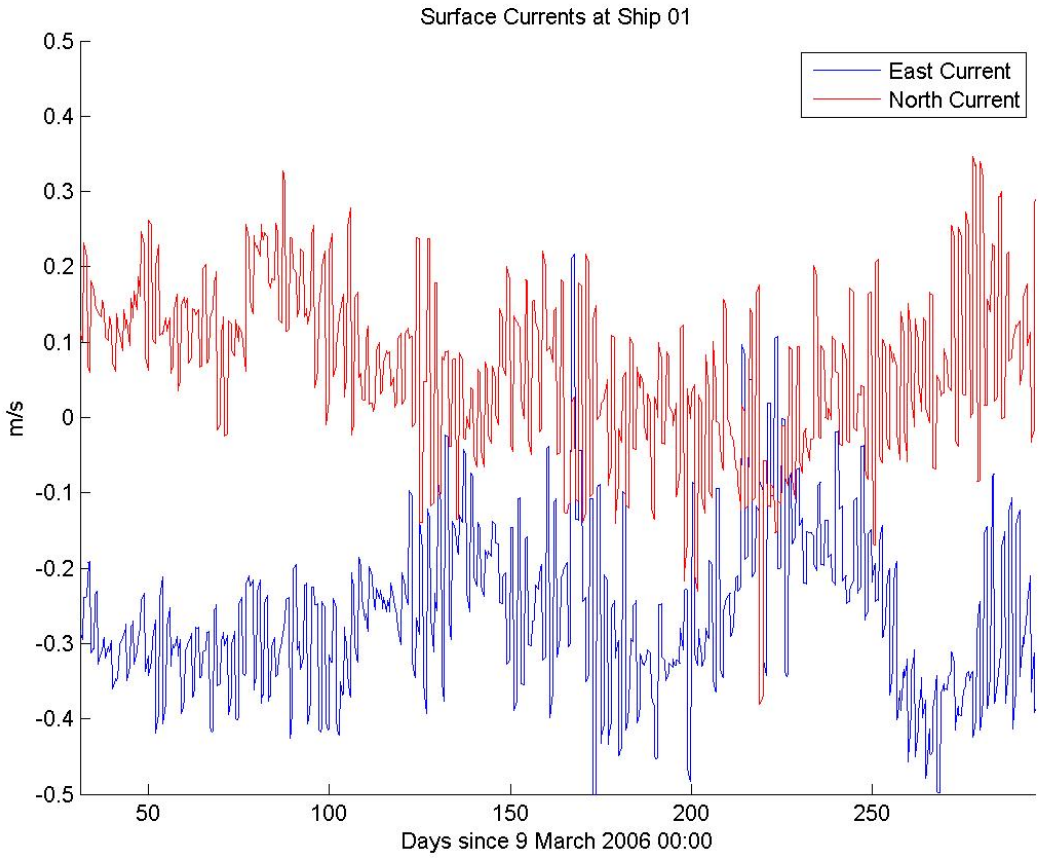


Figure 5.4. Time series of NCOM 1/8° global surface currents at 188.943665° W, 12.304348° N.

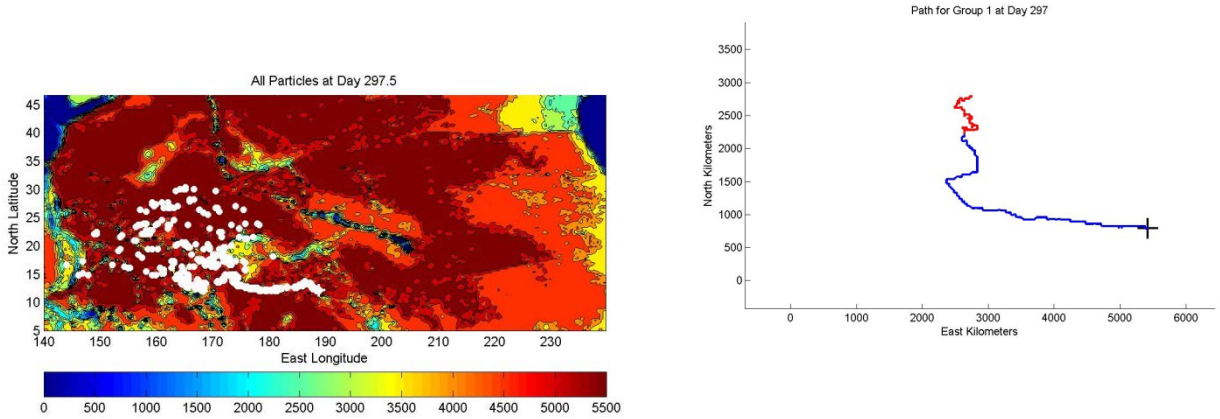


Figure 5.5. Particle tracking results for North Pacific with NCOM 1/8° database. **A.** All particles present in the simulation on Day 297.5 (31 Dec 2006 24:00) with a source in the south-central Pacific. **B.** Path of one particle for the simulation from 3 Mar to 31 Dec 2006 (Group 1, particle 1).

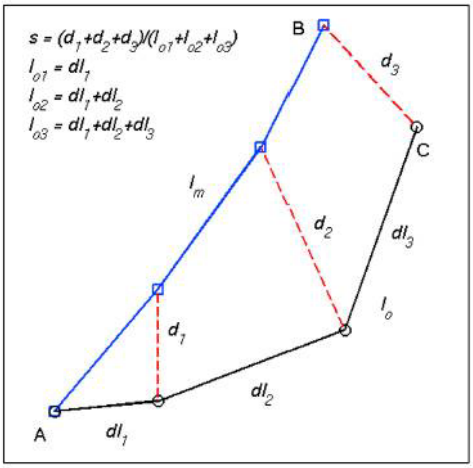


Figure 3. Illustration of the separation distances (d) between modeled and observed endpoints of Lagrangian trajectories (A-B and A-C, respectively). The lengths of the modeled and observed trajectories between the start- and end-points are shown as l_m and l_o, respectively. An index is defined as an average of the separation distances weighted by the lengths of the observed trajectory: $s = \sum_{i=1}^{N} d_i \Big/ \sum_{i=1}^{N} l_{oi}$, where N is total number of time steps. This index is used to define a trajectory model skill score $ss = \begin{cases} 1 - \frac{s}{n}, & (s \le n) \\ 0, & (s > n) \end{cases}$, where n is a tolerance threshold.

Figure 5.**6.** Schematic of the definition of the separation distance calculation (Liu and Weisberg, 2011).

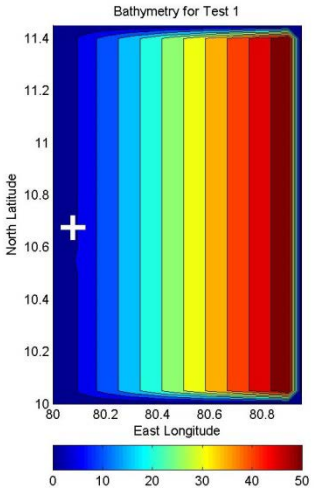


Figure 5.**7.** Test grid 1 bathymetry. The source is indicated by the white

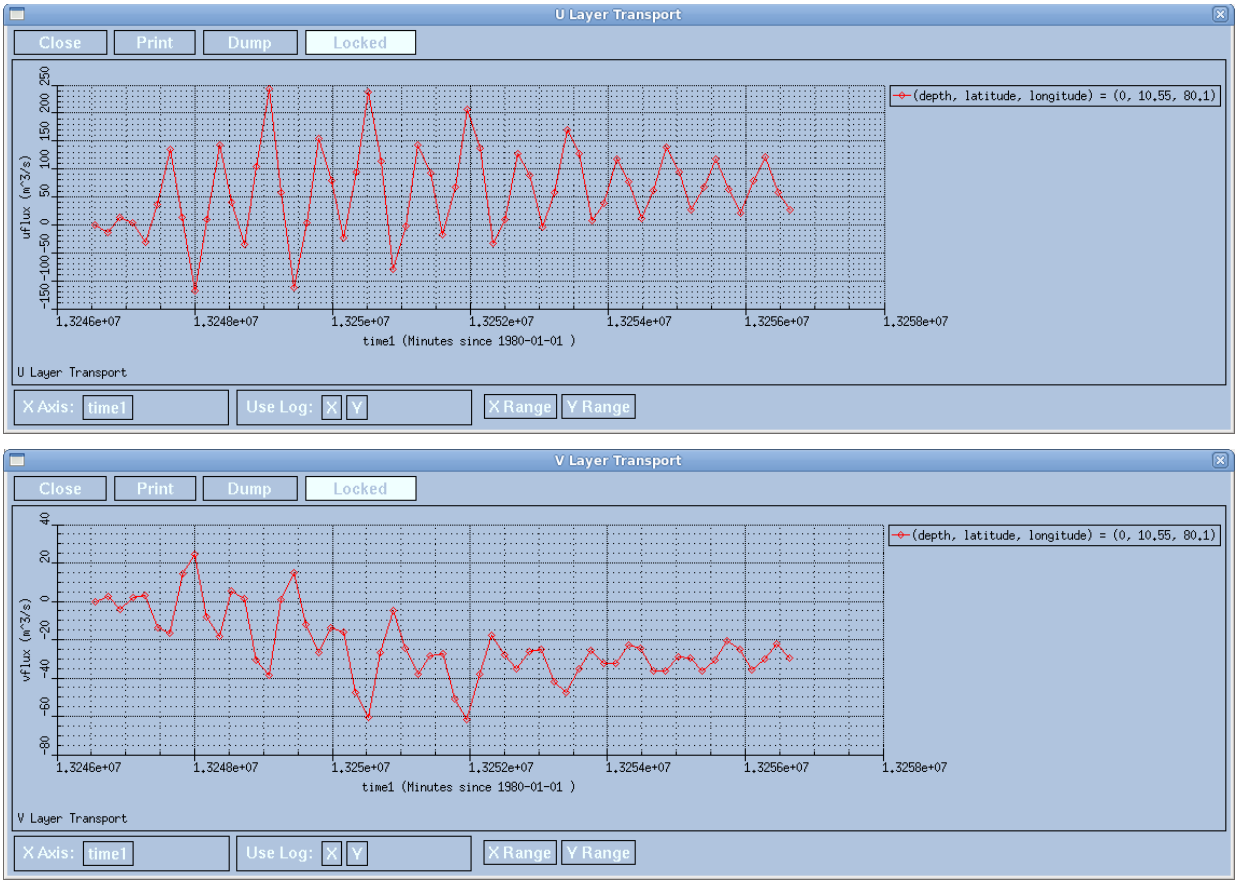


Figure 5.8. Surface transports from NCOM near the river mouth indicated by the cross in Figure 1. **A.** Eastward Positive. **B.** Northward Positive

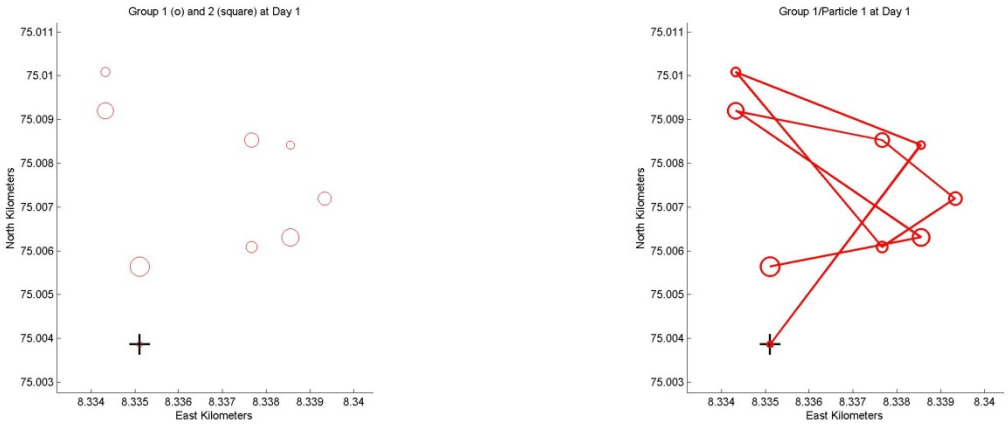


Figure 5.9. Particle locations after 24 hours for Case 1. **A.** All positions printed at 3 hr intervals. The circle diameter increases with age. **B.** Same as A but with the locations connected by a path. Each path segment is 3 hours long.

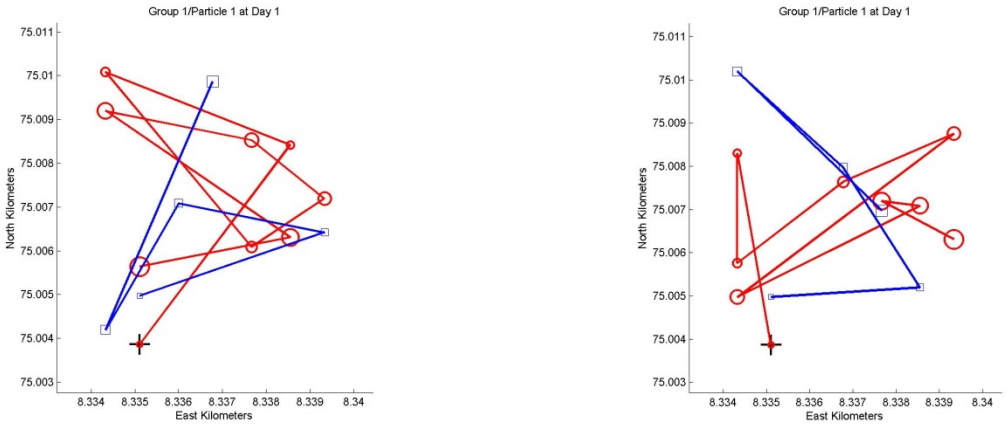


Figure 5.10. Particle locations after 24 hours for Case 1 with two groups. **A.** Same as Figure 3B but with the locations of group 2 indicated by blue boxes and a blue path. Each path segment is 3 hours long. **B.** Same as A but with *khcon* = 0.0001.

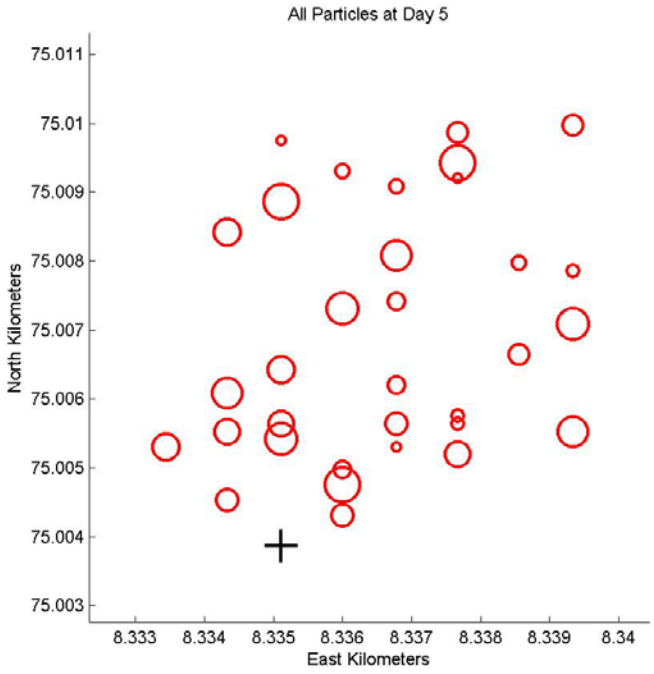


Figure 5.**11.** Final distribution of particles after 5 days. Both *horcon* and *khcon* = 0.0001. Older particles are smaller.

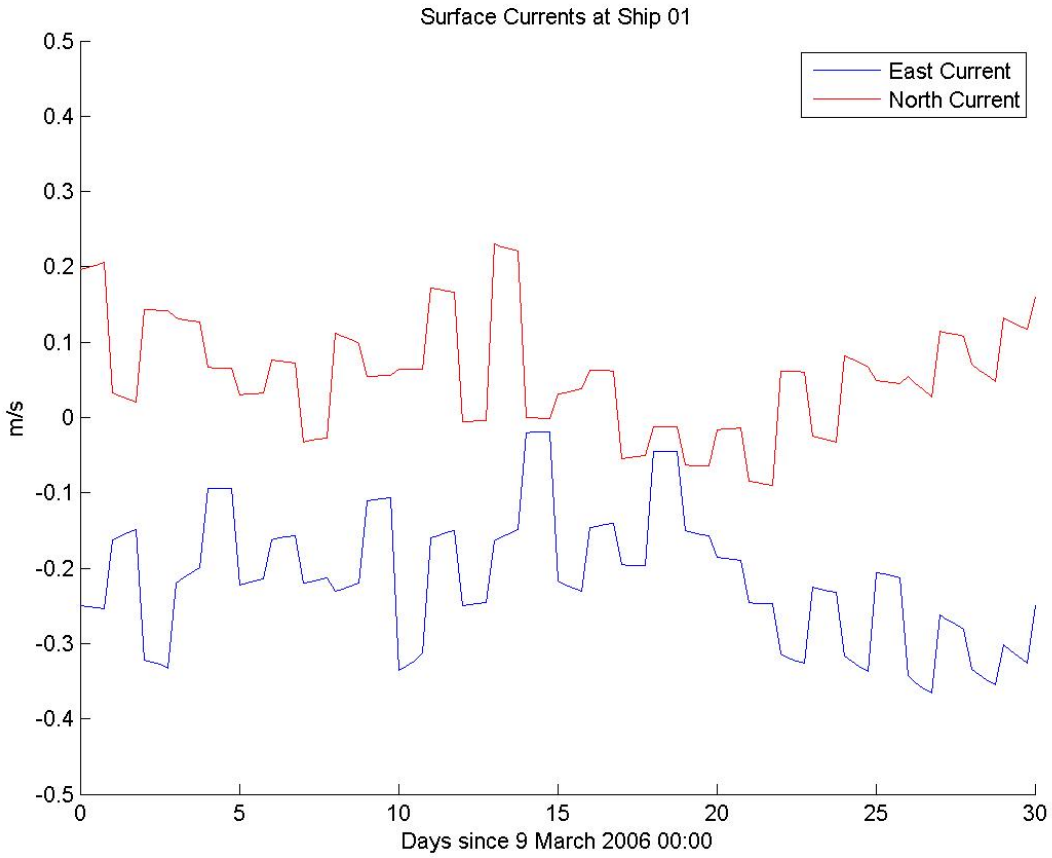


Figure 5.**12.** Surface currents interpolated to the location of particles indicated in Figure 6. The values are not interpolated in time, resulting in the jagged appearance.

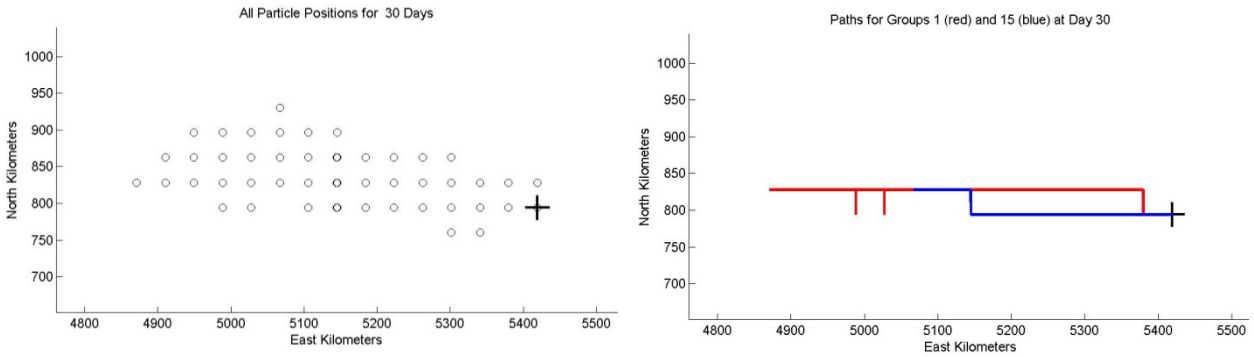


Figure 5.13. Particle locations after 30 days for Case 2. **A.** All positions printed at 12 hr intervals. **B.** Same as A but with the locations connected by a path. Each path segment is 12 hours long.

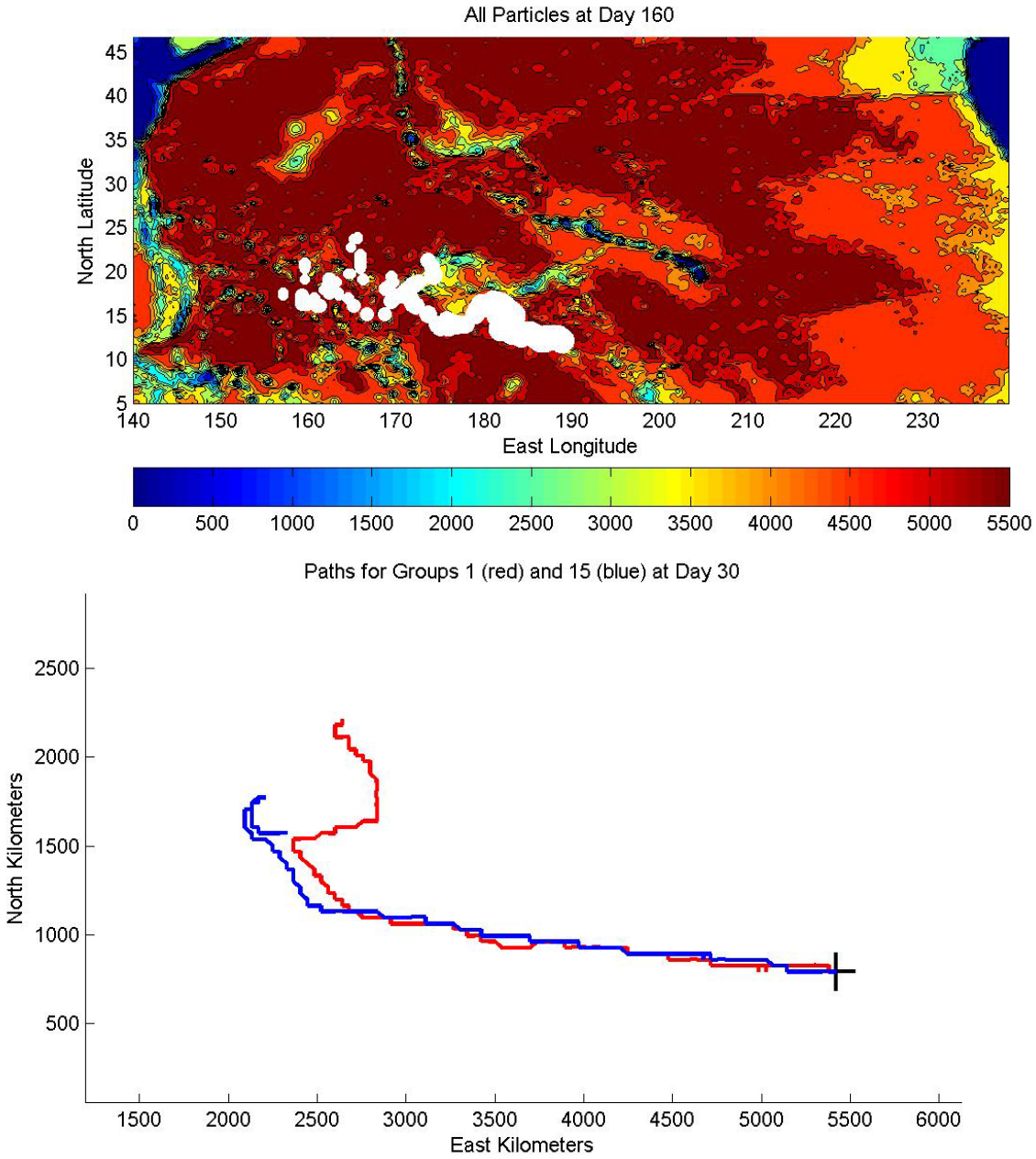


Figure 5.14. Results for 180 day simulation of Case 2. **A.** All final positions at day 160. **B.** Tracks of group 1 (day 0) and group 15 (day 15) for the entire simulation.

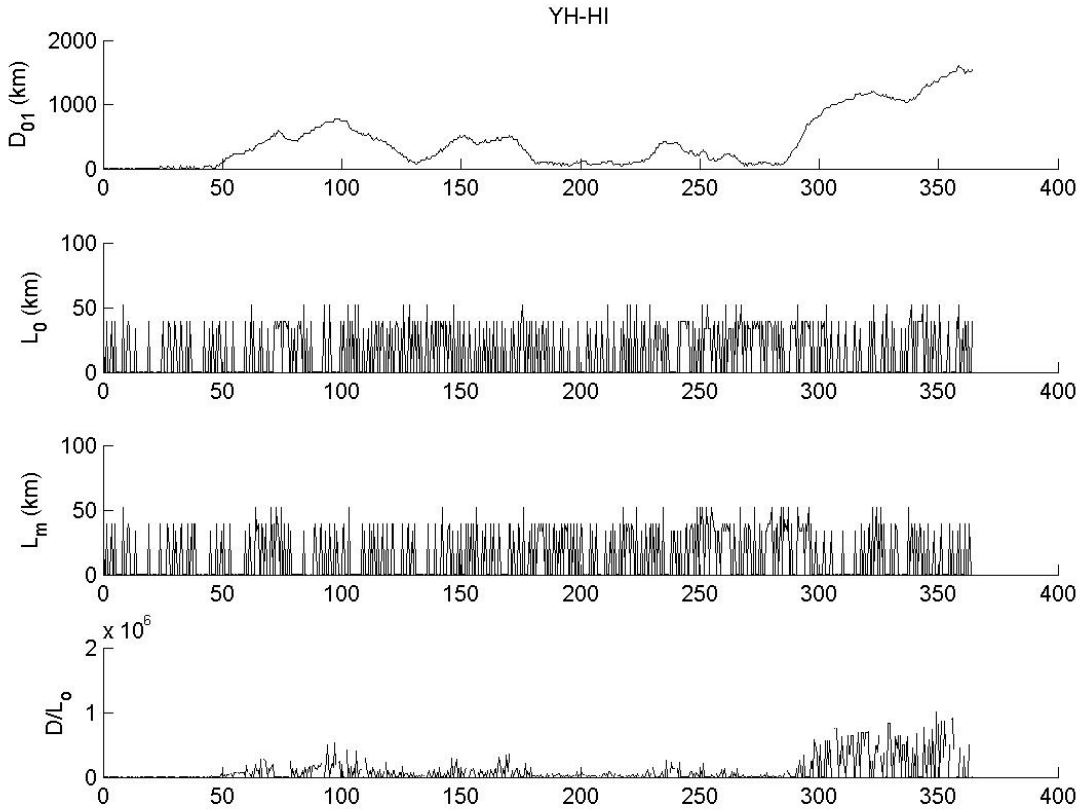


Figure 5.15. Path analysis for an experiment using the Smagorinsky mixing with *smagcon* = 0.1 compared to baseline at 12 hour intervals. (A) Separation distance; (B) and (C) path trajectory lengths for baseline and test, respectively; and (D) normalized separation distance using the baseline path length.

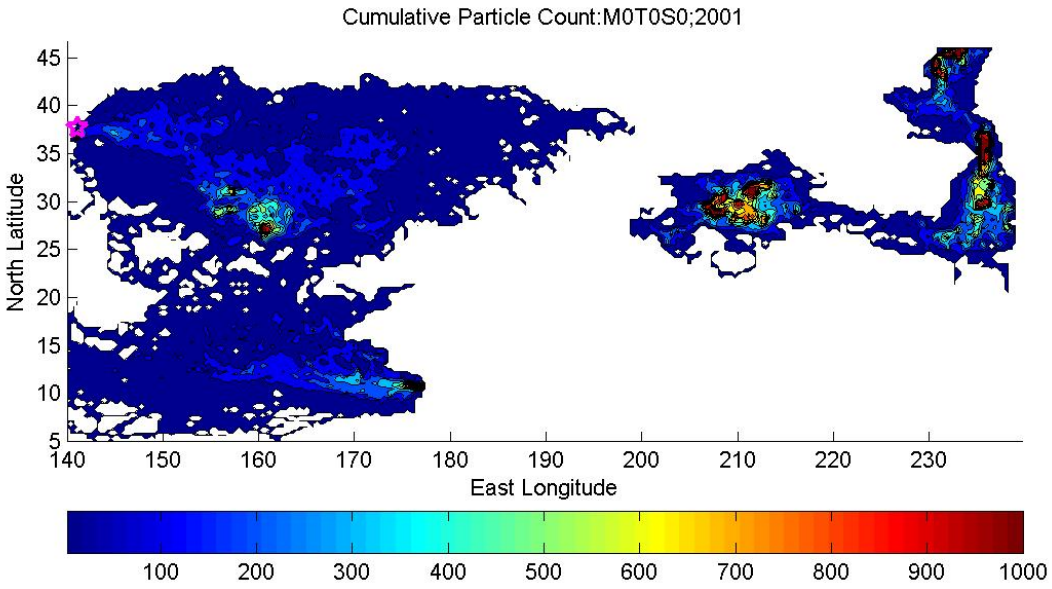


Figure 5.**16.** Baseline model cumulative particle count. This plot includes all particle locations for 2001 at time steps of 12 hours. Only one particle was used because there is no mixing.

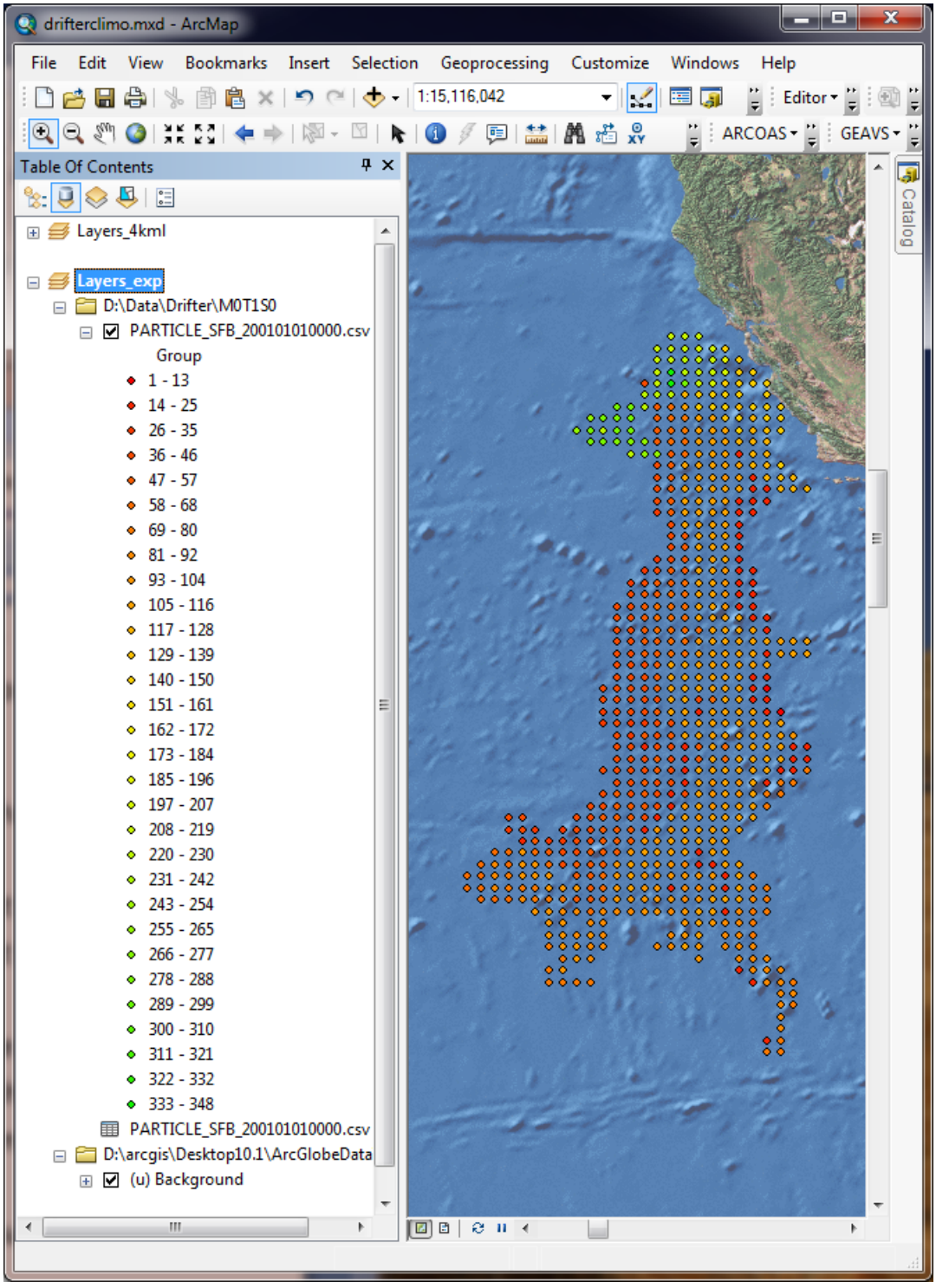


Figure 5.17. Points sampled in Figure 1 from the example data set of drifters seeded at San Francisco Bay. Points are coloured by group, the latest being the most green.

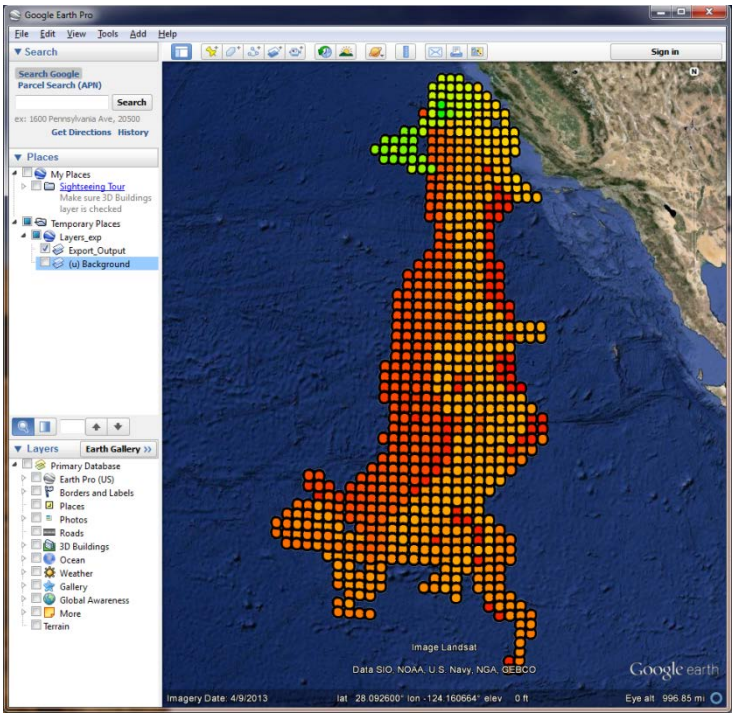


Figure 5.18. KMZ converted from ArcMap features on display in Google Earth.

References

Adala, A., and N. Tabbane. 2010. Discovery of semantic Web Services with an enhanced-Chord-based P2P network. International Journal of Communication Systems **23:** 1353-1365.

Allen, G., Bogden, P., Creager, G., Dekate, C., Jesch, C., Kaiser, H., McLaren, J., Perrie, W., Stone, G. W. and Zhang, X. 2008. Towards an integrated GIS-based coastal forecast workflow. Concurrency and Computation: Practice and Experience, **20:** 1637–1651

Almeida, J. M., M. A. Goncalves, F. Figueiredo, H. Pinto, and F. Belem. 2010. On the Quality of Information for Web 2.0 Services. Ieee Internet Computing **14:** 47-55.

Ankolekar, A., M. Krotzsch, T. Tran, and D. Vrandecic. 2008. The two cultures: Mashing up Web 2.0 and the Semantic Web. Journal of Web Semantics **6:** 70-75.

Anonymous. 2000. UDDI project. Online Information Review **24:** 465-465.

---. 2001. WSDL goes to W3C for standardization. Dr. Dobbs J. **26:** 18-18.

Battle, R., and E. Benson. 2008. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). Web Semantics: Science, Services and Agents on the World Wide Web **6:** 61-69.

Bennett, K. and others 2001. An architectural model for service-based software with ultra rapid evolution, p. 292-300. Ieee International Conference on Software Maintenance, Proceedings - Systems and Software Evolution in the Era of the Internet. Proceedings - Ieee International Conference on Software Maintenance.

Bertolino, A., and Acm. 2009. Approaches to Testing Service-Oriented Software Systems. Assoc Computing Machinery.

Bittencourt, M. L., E. G. Borges, E. F. Carvalho, and R. A. Augusto. 2008. An application service provider for finite element analysis. Advances in Engineering Software **39:** 899-910.

Boll, S. 2007. MultiTube - Where multimedia and Web 2.0 could meet. Ieee Multimedia **14:** 9-13.

Breivold, H. P., and M. Larsson. 2007. Component-based and service-oriented software engineering: Key concepts and principles, p. 13-20. *In* P. Muller, P. Liggesmeyer and E. Maehle [eds.], SEAA 2007: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, Proceedings. Euromicro Conference - Proceedings. Ieee Computer Soc.

Bulatovic, V., T. Ninkov, and Z. Susic. 2010. Open Geospatial Consortium Web Services in Complex Distribution Systems. Geod. List. **64:** 13-29.

Carone, T. E. 1996. Middleware and three-tier client/server development - Transaction managers are key to distributed-processing success. Dr. Dobbs J. **21:** 16-+.

Chakrabarti, S. K., P. Kumar, and Ieee. 2009. Test-the-REST: An Approach to Testing RESTful Web-Services. Ieee.

Cobb, M., T. R. Keen, and N. D. Walker. 2008. Modeling the circulation of the Atchafalaya Bay system during winter cold front events. Part 1: Model description and validation. Journal of Coastal Research **24:** 1036-1047.

Corno, G., Karl, D. M., Church, M. J., Letelier, R. M., Lukas, R., Bidigare, R. R., and Abbott, M. R. (2007), Impact of climate forcing on ecosystem processes in the North Pacific Subtropical Gyre. *J. Geophysical Res.*, 112 (C4), Art. C04021, doi:10.1029/2006JC003730.

Crnkovic, I., G. T. Heineman, H. W. Schmidt, J. Stafford, and K. Wallnau. 2007. Guest Editorial. Journal of Systems and Software **80:** 641-642.

Crnkovic, I., and M. Larsson. 2002. Challenges of component-based development. Journal of Systems and Software **61:** 201-212.

Crnkovic, I. 2001. Component-based software engineering — new challenges in software development. Software Focus **2**: 127–133.

Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. 2002. Unraveling the Web services Web - An introduction to SOAP, WSDL, and UDDI. Ieee Internet Computing **6:** 86-93.

Davidson, M. A., and E. Yoran. 2007. Enterprise security for web 2.0. Computer **40:** 117-119.

De Melo, A. C. V., and P. Silveira. 2010. Improving data perturbation testing techniques for Web services. Information Sciences **181:** 600-619.

Decker, S., P. Mitra, and S. Melnik. 2000. Framework for the semantic Web: an RDF tutorial. Internet Computing, IEEE **4:** 68-73.

Díaz, L., C. Granell and M. Gould, Case study: geospatial processing services for web-based hydrological applications. In: J.T. Sample, K. Shaw, S. Tu and M. Abdelguerfi, Editors, *Geospatial Services and Applications for the Internet*, Springer, New York (2008), pp. 31–47

Dietze, S., A. Gughotta, and J. Domingue. 2007. A semantic web services-based infrastructure for context-adaptive process support.

El-Bakry, H. M., A. M. Riad, Q. F. Hassan, A. E. Hassan, and N. Mastorakis. 2010. Technology and Recent Development of XML Web Services, p. 110-136. *In* L. A. Zadeh et al. [eds.], Recent Advances in Business Administration. Recent Advances in Computer Engineering. World Scientific and Engineering Acad and Soc.

England, P., R. Allen, and R. Underwood. 1996. RAVE: Real-time services for the Web. Comput. Netw. ISDN Syst. **28:** 1547-1558.

Felber, P., C. Y. Chan, M. Garofalakis, and R. Rastogi. 2003. Scalable filtering of XML data for Web services. Ieee Internet Computing **7:** 49-57.

Fielding, R. T., and R. N. Taylor. 2002. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology **2:** 115-150.

Fountain, T. C. 2003. Web Service Oriented Architecture: "Smart Operations" and IT strategy. C S R E a Press.

Gordon, A. D. 2002. XML Web Services: The global computer?, p. 355-355. *In* R. BaezaYates, U. Montanari and N. Santoro [eds.], Foundations of Information Technology in the Era of Network and Mobile Computing. International Federation for Information Processing. Kluwer Academic Publishers.

Greaves, M., and P. Mika. 2008. Semantic Web and Web 2.0. Web Semantics: Science, Services and Agents on the World Wide Web **6:** 1-3.

Hellman, R. 1999. A semantic approach adds meaning to the Web. Computer **32:** 13-16.

Huhns, M. N., and M. P. Singh. 2005. Service-oriented computing: Key concepts and principles. Ieee Internet Computing **9:** 75-81.

Keen, T. R., Furukawa, Y., Bentley, S. J., Slingerland, R. L., Teague, W. J., Dykes, J. D., and Rowley, C. D. (2006). Geological and oceanographic perspectives on event bed formation during Hurricane Katrina. *Geophys. Res. Let.*, 33, Paper L23614.

Keen, T. R., and K. T. Holland. 2010. The coastal dynamics of heterogeneous sedimentary environments: Numerical modeling of hydrodynamics and mass transport in estuaries, p. 132. Naval Research Laboratory.

Khare, R., F. N. Taylor, and S. Ieee Computer. 2004. Extending the REpresentational State Transfer (REST) architectural style for decentralized systems, p. 428-437. Icse 2004: 26th International Conference on Software Engineering, Proceedings. International Conference on Software Engineering.

Kona, S., A. Bansal, and G. Gupta. 2007. Automatic composition of semantic Web services.

Kono, N. and others 2009. Pathway Projector: Web-Based Zoomable Pathway Browser Using KEGG Atlas and Google Maps API. Plos One **4:** A47-A56.

Kovac, D., and D. Trcek. 2009. Qualitative trust modeling in SOA. Journal of Systems Architecture **55:** 255-263.

Kumar, M. 1988. WORLD GEODETIC SYSTEM 1984 - A MODERN AND ACCURATE GLOBAL REFERENCE FRAME. Marine Geodesy **12:** 117-126.

Kwon, Y., Y. Shigemoto, Y. Kuwana, and H. Sugawara. 2009. Web API for biology with a workflow navigation system. Nucleic Acids Research **37:** W11-W16.

Leathrum, J. F., and K. A. Liburdy. 1995. Formal test specifications in IEEE POSIX. Computer Standards & Interfaces **17:** 603-614.

Li, H. H., X. Y. Du, and X. A. Tian. 2007. A WSMO-Based semantic web services discovery framework in heterogeneous ontologies environment, p. 617-622. *In* Z. Zhang and J. Siekmann [eds.], Knowledge Science, Engineering and Management. Lecture Notes in Artificial Intelligence. Springer-Verlag Berlin.

Li, J. X., D. C. Zhang, J. P. Huai, and J. Xu. 2009. Context-aware trust negotiation in peer-to-peer service collaborations. Peer-to-Peer Networking and Applications **2:** 164-177.

Lin, K. J. 2007. Building Web 2.0. Computer **40:** 101-102.

Liu, B., M. H. Shi, and D. R. Chen. 2001. The simple object access protocol in Web Services architecture. International Academic Publishers Ltd.

Maaref, B., S. Nasri, and P. Sicard. 1997. Application programming interface framework for distributed environment based on object-oriented communications. Soc Computer Simulation.

Malek, S., H. Ramnath Krishnan, and J. Srinivasan. 2010. Enhancing middleware support for architecture-based development through compositional weaving of styles. Journal of Systems and Software **83:** 2513-2527.

Mccreedy, F. P., D. B. Marks, and Ieee. 2009. The Naval Research Laboratory's Ongoing Implementation of the Open Geospatial Consortium's Catalogue Services Specification, p. 2796-2802. Oceans 2009, Vols 1-3. Oceans-Ieee. Ieee.

Mclellan, S. G., A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi. 1998. Building more usable APIs. Ieee Software **15:** 78-+.

Medjahed, B., and A. Bouguettaya. 2005. A dynamic foundational architecture for semantic Web services. Distributed and Parallel Databases **17:** 179-206.

Mezini, M., and K. Lieberherr. 1998. Adaptive plug-and-play components for evolutionary software development. ACM Sigplan Not. **33:** 97-116.

Michelsen, J., and Ieee. 2007. For Net-Centric operations, the future of quality is federated: Establishing horizontal trust in federated SOA application environments, p. 3133-3140. 2007 Ieee Military Communications Conference, Vols 1-8. IEEE Military Communications Conference.

Moore, C. (2003), Trashed--Across the Pacific Ocean, plastics, plastics, everywhere. *Natural History*, 112 (9), 46-51.

Nachouki, G., and M. Quafafou. 2011. MashUp web data sources and services based on semantic queries. Information Systems **36:** 151-173.

Osipov, M. A., O. L. Machul'sky, and L. A. Kalinichenko. 2000. Mapping the XML data model into the object model of the SYNTHESIS language. Programming and Computer Software **26:** 192-198.

Pais, V. F., S. Balme, H. S. Akpangny, F. Iannone, and P. Strand. 2010. Enabling remote access to projects in a large collaborative environment. Fusion Engineering and Design **85:** 633-636.

Papazoglou, M. P., and W. J. Van Den Heuvel. 2007. Service oriented architectures: approaches, technologies and research issues. Vldb Journal **16:** 389-415.

Parastatidis, S., J. Webber, P. Watson, and T. Rischbeck. 2005. WS-GAF: a framework for building Grid applications using Web Services. Concurrency and Computation-Practice & Experience **17:** 391-417.

Peltz, C. 2003. Web services orchestration and choreography. Computer **36:** 46-+.

Poo, G. S., and C. G. Chew. 1996. Modeling of the XOM/XMP application programming interface (API). Ieee Communications Magazine **34:** 134-144.

Potmesil, M., 1997. Maps alive: viewing geospatial information on the WWW, Computer Networks and ISDN Systems, Volume 29, Issues 8-13, Papers from the Sixth International World Wide Web Conference, September 1997, Pages 1327-1342

Pullen, J. M. and others 2005. Using Web services to integrate heterogeneous simulations in a grid environment. Future Generation Computer Systems **21:** 97-106.

Putz, S. 1994. INTERACTIVE INFORMATION-SERVICES USING WORLDWIDE WEB HYPERTEXT. Comput. Netw. ISDN Syst. **27:** 273-280.

Qu, Z. Y., Y. T. Ge, K. Y. Jiang, and T. G. Lu. 2007. Key Issues in Building Web-based Services. Ieee Computer Soc.

Ravat, F., O. Teste, R. Tournier, and G. Zurfluh. 2010. Finding an application-appropriate model for XML data warehouses. Information Systems **35:** 662-687.

Rezende, E. S., E. Dodonov, R. S. Ulson, M. A. Cavenaghi, and R. S. Lobato. 2009. Towards Interoperability in P2P World: an Indexing Middleware for Multi-Protocol Peer-to-Peer Data Sharing.

Sacha, J. and others 2010. Decentralising a service-oriented architecture. Peer-to-Peer Networking and Applications **3:** 323-350.

Sample, John T., E Ioup, 2010. Tile-Based Geospatial Information Systems: Principles and Practices, ISBN 978-1-4419-7630-7. Springer Science+Business Media, LLC, 2010

Severance, C. 1999. Posix: A model for future computing. Computer **32:** 131-132.

Shishir Gundavaram, S., 1996. CGI Programming on the World Wide Web, 433 pages, First Edition, March 1996.

Sousa, P., and V. Freitas. 1998. A framework for the development of tolerant real-time applications. Comput. Netw. ISDN Syst. **30:** 1531-1541.

Staab, S. and others 2000. Semantic community Web portals. Computer Networks-the International Journal of Computer and Telecommunications Networking **33:** 473-491.

Stojanovic, Z., A. Dahanayake, H. Sol, and Ieee. 2004. Modeling and design of service-oriented architecture, p. 4147-4152. 2004 Ieee International Conference on Systems, Man & Cybernetics, Vols 1-7. Ieee International Conference on Systems, Man, and Cybernetics, Conference Proceedings. Ieee.

Trillo, R., L. Po, S. Ilarri, S. Bergamaschi, and E. Mena. 2010. Using semantic techniques to access web data. Information Systems **36:** 117-133.

True, S. A., and Ieee. 2004. Planning the future of the world geodetic system 1984.

Tsai, W. T. 2005. Service-Oriented System Engineering: A new paradigm. Ieee Computer Soc.

Vandervoort, H. 1992. MIDDLEWARE - A CHECKLIST FOR MIGRATING TO DISTRIBUTED COMPUTING. Morgan Kaufmann Pub Inc.

Villoldo, E. J., J. Serrat-Fernandez, and E. Luque. 2007. Improving Web Services Interoperability with Binding Extensions, p. 873-879. Web Services, 2007. ICWS 2007. IEEE International Conference on.

Wang, M., B. Y. Zhang, and J. W. Zhang. 2001. BES Monitoring & Displaying System. Science Press.

Weis, F. 1996. Design of an object MMS Application Programming Interface. International Society Computer S & Their Applications (Isca).

White, W. B., and Hasunuma, K. (1980), Interannual variability in the baroclinic gyre structure of the western North Pacific from 1954-1974. *J. Marine Res.*, 38 (4), 651-672.

Yu, J., B. Benatallah, F. Casati, and F. Daniel. 2008. Understanding mashup development. Ieee Internet Computing **12:** 44-52.